Technical Report of the project:
Training on Environmental Modelling and Applications (TEMA)

CdC 5871

Learning by doing on the EGEE GRID and first performance analysis of CODESA-3D multirun submission

Fabrizio Murgia

(Energy & Environment Program, Environmental Sciences Group)

October 2006

Center for Advanced Studies, Research and Development in Sardinia (CRS4)
Scientific & Technological Park, POLARIS, Edificio 1, 09010 PULA (CA - Italy)

Preface

The project TEMA (Training on Environmental Modelling and Applications) is a CRS4 training initiative in the field of computational hydrology and grid computing (Jan-Sept, 2006). The personnel involved were Fabrizio Murgia (trainee) and Giuditta Lecca (tutor).

The objectives of the project were:

- To aquire specialized skills about grid computing with special emphasis on computational sub-surface hydrology;
- To develop and test software procedures to run Monte Carlo simulations on the EGEE production grid;
- To produce a technical report and some seminars about grid computing.

The aquired competences and skills will be used in the ongoing projects GRIDA3, CyberSAR and DEGREE.

1 Grid general notes

1.1 Introduction

Grid is an infrastructure that involves the integrated and collaborative use of computers, networks, databases and scientific instruments owned and managed by multiple organizations. Grid applications often involve large amounts of data and/or computing resources that require secure resource sharing across organizational boundaries. This makes Grid application management and deployment a complex undertaking, whose goal is to provide a service-oriented infrastructure that leverages standardized protocols and services to enable pervasive access to and coordinate sharing of geographically distributed hardware, software, and information resources [PAR05]. Grid middlewares provide users with seamless computing ability and uniform access to resources in the heterogeneous Grid environment. Several software toolkits and systems have been developed, all over the world [BUY05]

1.2 Grid little history

Everybody regards the electricity as coming from the "National Grid" which is an abstraction allowing users of electrical energy to gain access to power from a range of different generating sources via a distribution network. A large number of different appliances can be driven by energy from the National Grid – table lamps, vacuum cleaners, washing machines, etc. – but they all have a simple interface, typically this is an electrical socket, to the National Grid [WAL02].
The concept of "computing utility" providing "continuous operation analogous to power and telephone" can be traced back to the Multics Project in 1960s. [COR65].
The term "the Grid" was coined in the mid1990s to denote a proposed distributed computing infrastructure for advanced science and engineering [FOS03], this was sometimes called "Metacomputing" [CAT92].
The Grid is an abstraction allowing transparent and pervasive access to distributed computing resources. Other desirable features of the Grid are that the access provided should be secure, dependable, efficient, and inexpensive, and enable a high degree of portability for computing applications.

1.3 Grid Virtual Organization

The real and specific problem that underlies the Grid concept is coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations. The sharing that we are concerned with is not primarily file exchange but rather direct access to computers, software, data, and other resources, as is required by a range of collaborative problem-solving and resource-brokering strategies emerging in industry, science, and engineering. This sharing is, necessarily, highly controlled, with resource providers and consumers defining clearly and carefully just what is shared, who is allowed to share, and the conditions under which sharing occurs. A set of individuals and/or institutions defined by such sharing rules form what we call a Virtual Organization (VO) [FOS01]. VOs vary tremendously in their purpose, scope, size, duration, structure, community, and sociology. Nevertheless, careful study of underlying technology requirements leads us to identify a broad set of common concerns and requirements.
There is a need:

- for highly flexible sharing relationships;
- for sharing of varied resources;
- for sophisticated and precise levels of control over how shared resources are used;
- for diverse usage modes;

The establishment, management, and exploitation of dynamic, cross-organizational VO sharing relationships require new technology. Effective VO operation requires establishing sharing relationships among any potential participants. Grid architecture is first and foremost a protocol architecture, with protocols defining the basic mechanisms by which VO users and resources negotiate, establish, manage, and exploit sharing relationships.
An important issue is our need to ensure that sharing relationships can be initiated among arbitrary parties, accommodating new participants dynamically, across different platforms, languages, and programming environments. Interoperability is thus the central issue to be addressed.
Without interoperability VO applications and participants are forced to enter into bilateral sharing arrangements, as there is not assurance that the mechanism used between any two parties will extend to any other parties. A solution could be to introduce a common horizontal layer that defines and implements a consistent set of abstractions and interfaces for access to, and management of, shared resources. We refer to this horizontal resource integration layer as "grid infrastructure".
A standards-based open architecture facilitates extensibility, interoperability, portability, and code sharing; these features constitute what is often termed middleware: "the services needed to support a common set of applications in a distributed network environment" [AIK00]. This is what enables the horizontal integration across diverse physical resources that we require to decouple application and hardware.
The real "innovation" in grid comes from the combination of technology domains that include workload virtualization, information virtualization, system virtualization, storage virtualization, provisioning, and orchestration. From this statement, one may already conclude that no single technology constitutes a grid, but, instead, the method with which broad sets of resources are accessed and combined. Grid computing is not about a specific hardware platform, a database or a particular piece of job management software, but the way in which IT resources dynamically interact to address changing business requirements [BER02].

1.4 Grid technical capabilities

A grid infrastructure must provide a set of technical capabilities, as follows:

- **Resource modelling**: Describes available resources, their capabilities, and the relationships between them to facilitate discovery, provisioning, and quality of service management.
- **Monitoring** and **Notification**: Provide visibility into the state of resources — and notifies applications and infrastructure management services of changes in state — to enable discovery and maintain quality of service. Logging of significant events and state transitions is also needed to support accounting and auditing functions.
- **Allocation**: Assures quality of service across an entire set of resources for the lifetime of their use by an application. This is enabled by negotiating the required level(s) of service and ensuring the availability of appropriate resources through some form of reservation-essentially, the dynamic creation of a service-level agreement.
- **Provisioning**, **life-cycle management**, and **decommissioning**: Enables an allocated resource to be configured automatically for application use, manage the resource for the duration of the task at hand, and restore the resource to its original state for future use.

- **Accounting** and **Auditing**: Tracks the usage of shared resources and provides mechanisms for transferring cost among user communities and for charging for resource use by applications and users.

A grid infrastructure must furthermore be structured so that the interfaces by which it provides access to these capabilities are formulated in terms of equivalent abstractions for different classes of components.

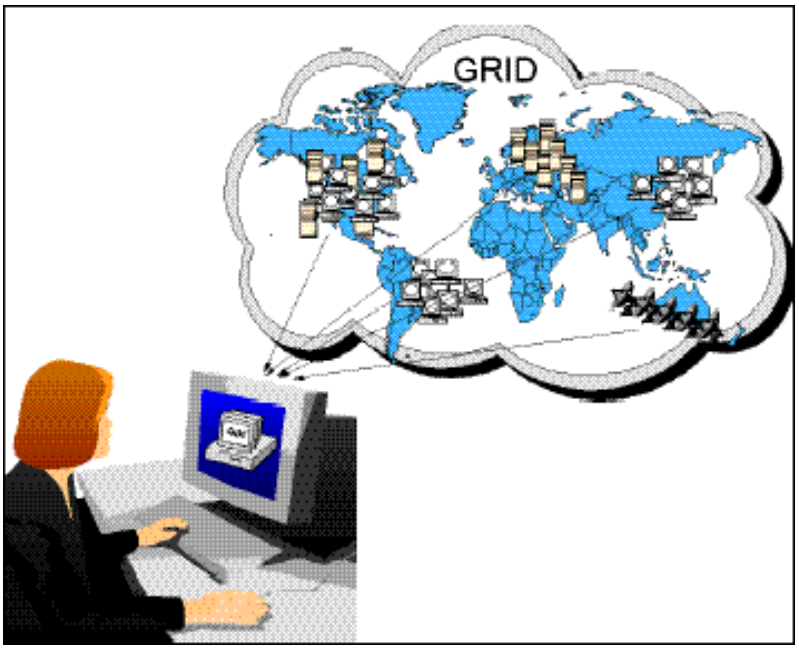


Figure 1: The complexity of grid infrastructure is transparent to the user.

For example, a client should be able to use the same authorization and quality-of-service negotiation operations when accessing a storage system, network, and computational resource; the user should see the grid as a single big computer (Figure 1). Without this uniformity, it becomes difficult for workload managers and other management systems to combine collections of resources effectively and automatically for use by applications. These considerations make the definition of an effective grid infrastructure a challenging task. Many of the standards and software systems needed to realize this goal, however, are already in place.

An effective grid infrastructure must implement management capabilities in a uniform manner across diverse resource types - and, if we are to avoid vendor lock-in, it should do so in a manner that does not involve commitment to any proprietary technology.

As was the case with the Internet and Web, both standards and open source software have important roles to play in achieving these goals. Standards enable interoperability among different vendor products, while open source software allows enterprises to proceed with deployments now, before all standards are available. Standards such as the Open Grid Services Architecture (OGSA [FOS02]) and tools such as those provided by the Globus Toolkit provide the necessary framework [JAC06].

In addition to CPU and storage resources, a grid can provide access to increased quantities of other resources and to special equipment, software, licenses, and other services. For example, increased bandwidth to the Internet to implement a data mining search engine or licensed software installed that the user requires or special devices. All of these will make the grid look like a large virtual machine with a collection of virtual resources beyond what would be available on just one conventional machine [FER03].

1.5 Grid goals

The current vision of grid computing is of uniform and controlled access to computing resources, seamless global aggregation of resources enabling seamless composition of services, and leading to autonomic self-managing behaviors.

1) **Seamless Aggregation of Resources and Services**: aggregation include both the aggregation of capacity, for instance clustering of individual system to increase computational power and storage capacity, as well as the aggregation of capability, for instance combining specialized instruments with large storage systems and computing clustering. Key capabilities include protocols and mechanisms to secure discovery, access to, aggregation of resources, development of applications.
2) **Ubiquitous Service-Oriented Architecture**: machines large and small and the services that they provide could be dynamically combined in a spectrum of VOs according to the needs and requirements of the participants involved.
3) **Autonomic Behaviors**: the emerging vision of above aims at realizing computing systems and applications capable of configuring, managing, interacting, optimizing, securing, and healing themselves with minimum human intervention, leading to a research initiatives such as Autonomic Grids [PAT03], Knowledge Grids [ZHU04], Cognitive or Semantic Grids [GEL04].

At the current time several basic Grid tools are stabilizing and many Grid projects are deploying sizable grids. They have to cope with some issues like:

- Creation of simple tools for reliable deployment, for use by nonspecialist and for supporting the enterprise-scale;
- Creation and validation of standards universal accepted (OGSI – Open Grid Services Infrastructure [TUE02] provides a uniform architecture for building and managing Grids and Grid Applications), moving towards the convergence of Web and Grid services Architecture;
- Overcome of non technical barriers, since Grid computing is about resource sharing and many organizations are fundamentally opposed to this.

1.6 Grid types

Grids can be used in a variety of ways to address various kinds of application requirements.

- **Computational grid**: is a software infrastructure that facilitates solving large-scale problems by providing the mechanisms to access, aggregate, and manage the computer network-based infrastructure of science [JOH02]. A computational grid is focused on setting aside resources specifically for computing power, providing easy access to many different types of resources to enable users to pick and choose those required to achieve their intended objectives. In this type of grid, most of the machines are high-performance servers.
- **Data grid**: a data grid [ALL01] is responsible for housing and providing access to data across multiple organizations. Users are not concerned with where this data is located as long as they have access to the data. For example, you may have two universities doing life science research, each with unique data. A data grid would allow them to share their data, manage the data, and manage security issues such as who has access to what data.

- **Scavenging grid**: a scavenging grid [CAS02] is most commonly used with large numbers of desktop machines. Machines are scavenged for available CPU cycles and other resources. Owners of the desktop machines are usually given control over when their resources are available to participate in the grid.
- **Enterprise grid**: Various kinds of distributed systems operate today in the enterprise, each aimed at solving different kinds of problems. In a typical SME (Small – Midsize Enterprise), there are many resources which are generally under-utilised for long periods of time. Any entity that could be used to fulfill any user requirement could be defined as a "resource"; this includes compute power (CPU), data storage, applications, and services. An enterprise grid can be loosely defined as a distributed system that aims to dynamically aggregate and co-ordinate various resources across the enterprise and improve their utilisation such that there is an overall increase in productivity [NAD05].
  There is an evolution of the approach on GRID computing, from one homogeneous to one heterogeneous, distributed and loosely-coupled (Figure 2).

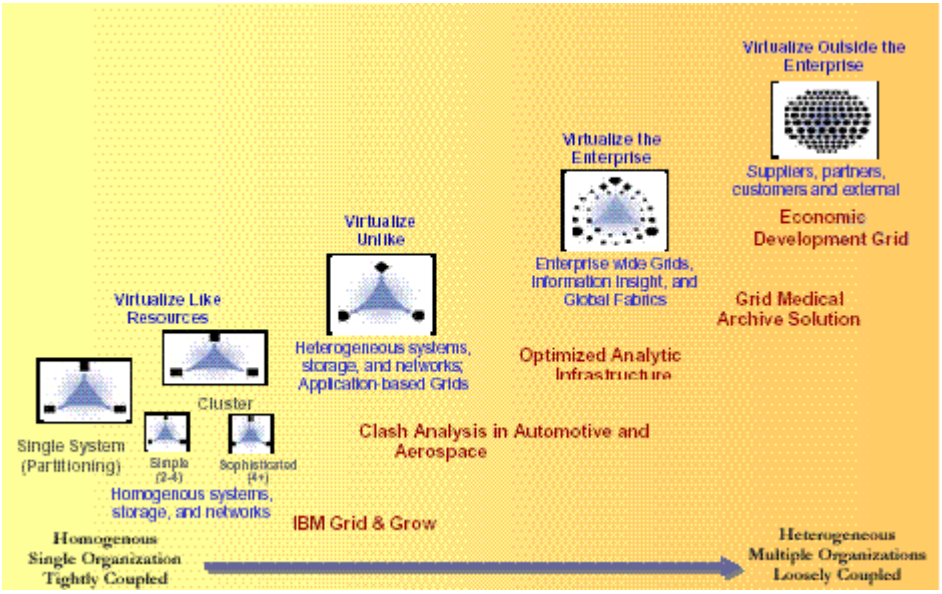


Figure 2: Grid Spectrum: as customers move from left to right on the spectrum, they are moving from a homogeneous environment within a single organization that is very tightly coupled to one that is heterogeneous, distributed and loosely-coupled.

1.7 Grid components

A grid is a distributed collection (Figure 3) of machines, sometimes referred to as "nodes", "resources", "members", "donors", "clients", "hosts", "engines" and many other such terms.

- **Computation**: most common resource is computing cycles provided by the processors of the machine on the grid;
  - **Jobs**: are programs that are executed at an appropriate point on the grid. They may compute something, execute one or more system commands, move or collect data, or operate machinery.
  - **Scheduling**: advanced grid systems include a job "scheduler" of some kind that automatically finds the most appropriate machine on which to run any given job that is waiting to be executed. The term "Resource Broker" is more used than scheduler.

- **Storage**: data storage can be memory attached to processor or hard disks or other permanent storage media. Storage capacity can be increased by using the storage on multiple machines with a unifying file system.
- **Communications**: meaning communications among machines within the grid and external to the grid.
- **Software**: some machines on grid may have software installed that may be too expensive to install on every grid machine. Within the grid, jobs requiring this software run just on the machines where it happens to be installed; this approach can save significant expenses for an organization.
- **Special devices**: platforms on the grid often have different architectures, operating systems, devices, capacities and equipment, representing a different kind of resources the grid can use.

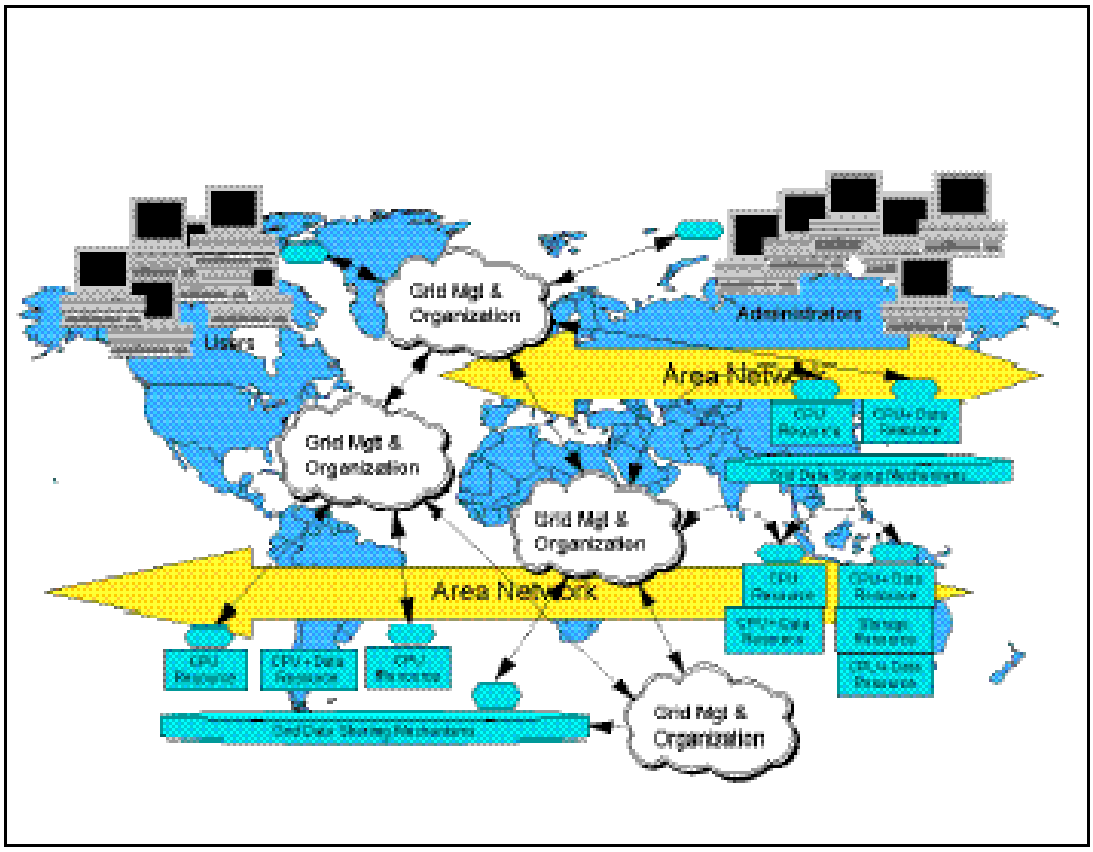


Figure 3: The First Big Idea behind the Grid is sharing of resources: entering the Grid to use remote resources, allows the user to do things that he cannot do with its own computer, or the computer centre he normally uses. This is more than simple file exchange: it is direct access to remote software, computers and data. It can even give access and control of remote sensors, telescopes and other devices. Resources are owned by many different people creating a situation amongst owners of computer resources where everyone concerned, sees the advantage of sharing, and there are mechanisms in place so that each resource provider feels they can trust any user who is trusted by any other resource provider.

Various Grid components satisfying above capabilities could be thought as arranged into layers, typically four layers (Figure 4):

- **Fabric Layer**: consists of distributed resources such as computers (multiple and various architectures running a variety of OO. SS.), network, storage devices and scientific instruments (sensors, telescopes etc.);
- **Core Middleware**: offers services such as remote process management, resources co-allocation, storage access, information registration, security, resource reservation and trading;

- **User – level Middleware**: using interfaces provided by the low-level middleware provides higher level abstractions and services. Are included application development environments, programming tools, resource brokers for managing resources, scheduling application tasks;
- **Applications and Portals**: for instance Web-enabled application services, where user can submit jobs and collect results via Web;

Each layer builds on the services offered by the lower layer in addition to interacting and co-operating with components at the same level.

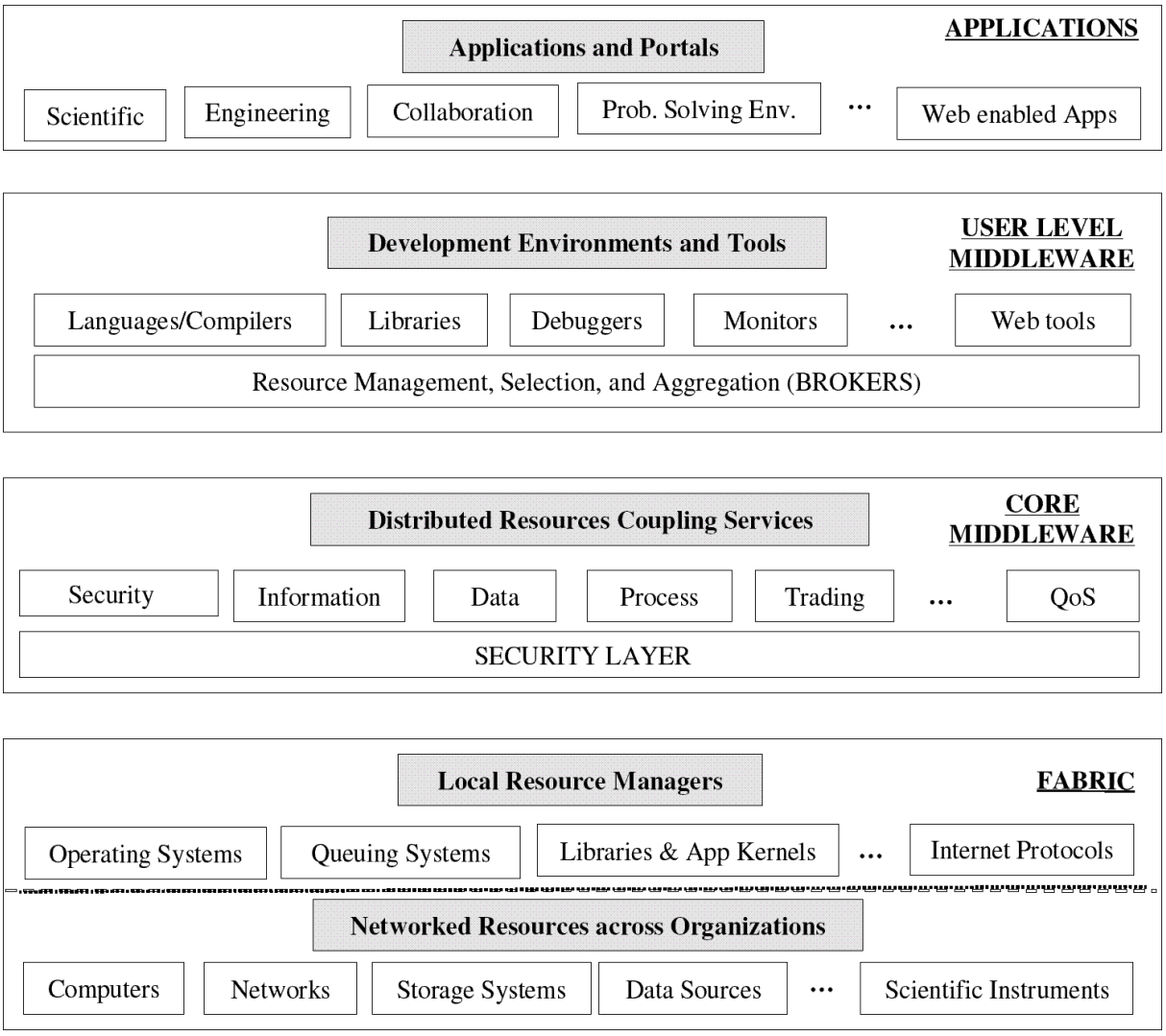


Figure 4: The architecture of the Grid is often described in terms of "layers", each providing a specific function. In general, the higher layers are focused on the user (user-centric), whereas the lower layers are more focused on computers and networks (hardware-centric).

1.8 Grid characteristics

1) **Large scale**:
   - the number of resources in a grid can be range from just a few to millions.
2) **Geographical distribution**:
   - resources may be located at distant places.

3) **Heterogeneity**:
    - hardware and software resources can be varied: different devices (super computers, PC, PDA, mobile phone, etc), different operating systems, etc.
4) **Multiple administrations**:
    - each virtual organization can establish different security policies under which their resources can be accessed and used.
5) **Resource sharing**:
    - users and organization contribute their resources to Grid.
6) **Resource coordination**:
    - coordinate use of resources to form an aggregate power.
7) **Transparent access**:
    - a grid is seen as a single virtual computer.
8) **Consistent access**:
    - standard services, protocol and interfaces to access resource from different grid's implementation.
9) **Predictable performance**:
    - A Grid should ensure several Quality of Service (QoS) requirements (e.g.: run time, availability of resources, etc). Even though, it is very difficult to achieve this characteristic in a heterogeneous environment because the lack of central control. Several fault-tolerance techniques can be used to solve this problem such as resubmit jobs, re-allocation when some resources are down during the job execution, redundant job execution (multiple copies of job can be run on different machines), etc.

2 GLOBUS, a core middleware

2.1 GLOBUS Toolkit

The Globus Toolkit (http://www.globus.org/toolkit/) is an open source toolkit, freely available in source code form for use by anyone, including both commercial and non-commercial purposes. Every application can have unique requirements, or even a small set of functions frequently recurring. Good-quality implementations of these functions can reduce development costs. Finding reusable solutions is the key to constructing infrastructure that can support many applications in the future. This process is iterative. Furthermore, if these implementations are widely adopted and/or implement standards, they can enhance interoperability. Globus software addresses both goals, using an open source model to encourage both contributions and adoption [FOS05], providing a basic infrastructure that can be used to construct portable, high performance implementations of a range of services.
Some examples of generic solutions that have been created through this process are the Globus Toolkit's security, information services, job management, and data transfer components.
Globus toolkit is a collection of tools composed by a set of modules constituting a Metacomputing Virtual Machine [FOS97]:

- Resource Location and Allocation
- Communications
  - Based on NEXUS communications library
- Unified Resource Information Services
  - Configuration details about resources
  - Instantaneous performance information
  - Application-specific information
- Authentication interface
  - Evolving toward GSS (Generic Security Service), that defines a standard procedure and API (Application Programming Interface) for obtaining credentials (passwords or certificates), for mutual authentication (client and server), and for message-oriented encryption and decryption. GSS is independent of any particular security mechanism.
- Process Creation
- Data Access

The Globus Toolkit aims to provide standard system components that can support a wide variety of highly customized applications and user interfaces without requiring a completely unique infrastructure to be developed for each application. Some standards used in GLOBUS Toolkit are:

- SSL/TLS v1 (from OpenSSL) (IETF - http://www.ietf.org/)
- LDAP v3 (from OpenLDAP) (IETF)
- X.509 Proxy Certificates (IETF)
- SOAP (W3C - http://www.w3.org/)
- HTTP (W3C)
- GridFTP v1.0 (GGF - http://www.gridforum.org/)
- OGSI v1.0 (GGF)

The Globus Toolkit doesn't provide a complete solution for Grid projects. The components in the Globus Toolkit (along with components from other sources) have to be organized and integrated according to a plan that fits the requirements of the rest of the project. The Globus Toolkit provides standard building blocks and tools for use by application developers and system integrators. These building blocks and tools have already been proven useful in other projects, necessary condition to be included in the Globus Toolkit.

Some of these blocks are:

- **Grid Security Infrastructure (GSI)**: provides secure communication and single sign-on authentication, with support for local control over access rights and mapping from global to local user identities;
- **Globus Resource Allocation Manager (GRAM)**: provides facilities resource allocation and process creation, monitoring, and management;
- **Grid File Transfer Protocol (GridFTP)**: implements a high-performance, secure data transfer mechanism based on an extension of the FTP protocol that allows parallel data transfer;
- **Monitoring and Discovery Service (MDS)**: provides a framework for publishing and accessing information about Grid resources;

Users and developers can access Globus services using both:

- Command-line tools;
- Application Programming Interfaces (APIs).

2.2 Security

2.2.1 Public Key Cryptography

Globus uses public key cryptography (also known as asymmetric cryptography) [DIF88] as the basis for its functionality. The primary motivations behind that choice are:

- The need for secure communication (authenticated and perhaps confidential) between elements of a computational Grid;
- The need to support security across organizational boundaries, thus prohibiting a centrally-managed security system;
- The need to support "single sign-on" for users of the Grid, including delegation of credentials for computations that involve multiple resources and/or sites;

Public key cryptography relies on two keys, known as "Public Key" and "Private Key". These keys are numbers (generally big prime numbers) that are mathematically related in such a way that if either key ("public"/"private") is used to encrypt a message, the other key ("private"/"public") must be used to decrypt it. Also important is the fact that it is nearly impossible (with our current knowledge of mathematics and available computing power) to obtain the second key from the first one and/or from any messages encoded with the first key. By making the public key available publicly and keeping private the private key, a person can prove that he or she holds the private key simply by encrypting a message. If the message can be decrypted using the public key, the person must have used the private key to encrypt the message. Using public key cryptography, it is possible to digitally "sign" a piece of information. Signing information essentially means assuring a recipient of the information that the information hasn't been tampered with since it left your hands.
To ensure these features, private keys, that are password encrypted, must be kept private!

2.2.2 Certificates

Certificates are a central concept on Grid security. Every user and service on the Grid is identified via a certificate, which contains necessary information to identify and authenticate the user or service. A GSI certificate includes four primary pieces of information:

1. A subject name, identifying the person or object that the certificate represents.
2. The public key belonging to the subject.
3. The identity of a Certification Authority (CA) that has signed the certificate to certify that the public key and the identity both belong to the subject.

4. The digital signature of the named CA.

Note that a third party (a CA) is used to certify the link between the public key and the subject in the certificate. GSI certificates are encoded in the X.509 certificate format (Figure 5), a standard data format for certificates established by the Internet Engineering Task Force (IETF) [http://www.ietf.org/]. GSI certificates allow mutual authentication, i.e. two parties certified by trusted CAs, can mutually recognize their attributes and permissions. Other characteristics are communication integrity and encryption.

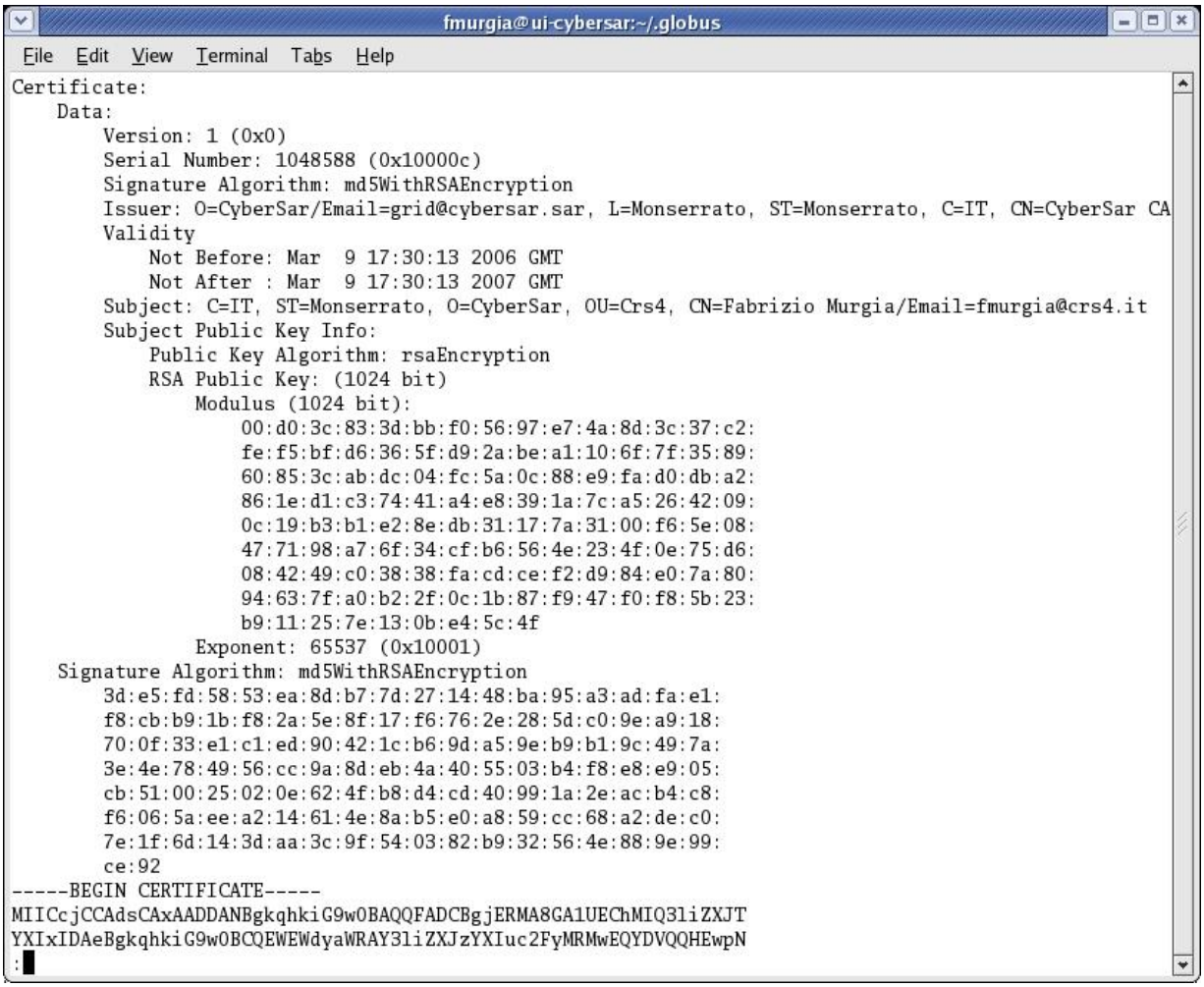


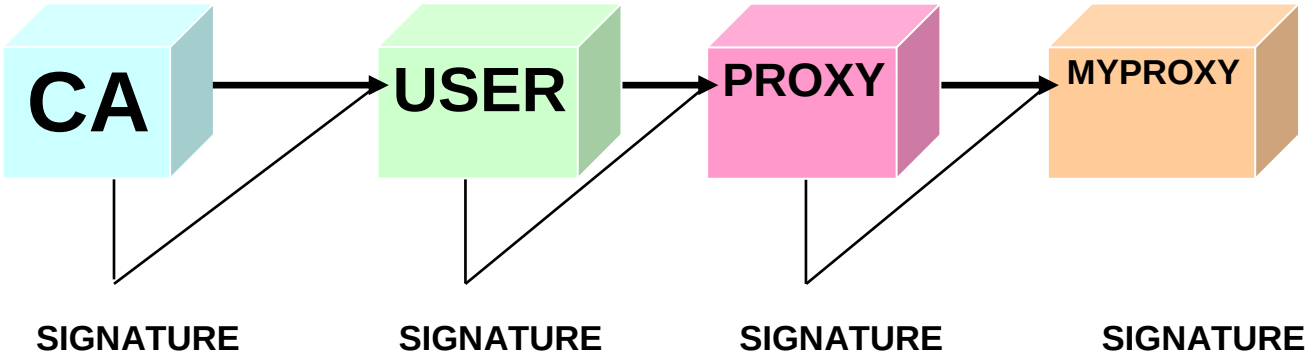
Figure 5: a certificate X.509 content as it appears.

2.2.3 Delegation, Single sign-on, proxy

Submitting a job on the grid can require that several Grid resources are used, in each one of these resources the user should be recognized and should login typing the private key passphrase. The need to re-enter the user's passphrase can be avoided by creating a *proxy*.

A proxy consists of a new certificate and a private key. The key pair that is used for the proxy, i.e. the public key embedded in the certificate and the private key, may either be regenerated for each proxy or obtained by other means. The new certificate contains the owner's identity, modified slightly to indicate that it is a proxy. The new certificate is signed by the owner, rather than a CA. (See diagram below.) The certificate also includes a time notation after which the proxy should no longer be accepted by others. To ensure a certain security level, proxies have limited lifetimes (default is 12 hours). The proxy pass-phrase must be kept secure too, but because the short validity of the proxy it does not need to be kept as secure as the owner's private key. Once a proxy is created and stored, the user can use the proxy certificate and private key for mutual authentication

without entering a password. When proxies are used, the remote party receives not only the proxy's certificate (signed by the owner), but also the owner's certificate. During mutual authentication, the owner's public key (obtained from his certificate) is used to validate the signature on the proxy certificate. The CA's public key is then used to validate the signature on the owner's certificate. This establishes a chain of trust from the CA to the proxy through the owner, as could be seen on the following schema.



2.2.3.1 Proxy Certificates useful commands:

- **glite-voms-proxy-init** (grid-proxy-init, edg-proxy-init, etc.), creates a proxy certificate. To create the certificate this tool asks the Grid Passphrase.

- **glite-voms-proxy-info** (grid-proxy-info, edg-proxy-info, etc.), visualizes information and status about the user's proxy.

- **glite-voms-proxy-destroy** (grid-proxy-destroy, edg-proxy-destroy, etc.), destroys the proxy certificate.

For options and other useful commands see Appendix Appendice A.

2.2.4 My Proxy Server

MyProxy [http://grid.ncsa.uiuc.edu/myproxy/] server is a daemon that runs on a trusted, secure host and manages a database of proxy credentials for use from remote sites. It is an open source software for managing X.509 Public Key Infrastructure (PKI) security credentials (certificates and private keys). MyProxy combines an online credential repository with an online certificate authority to allow users to securely obtain credentials when and where needed [NOV01]. Storing credentials in a MyProxy repository allows users to easily obtain proxy credentials, without worrying about managing private key and certificate files.
Users can use MyProxy to delegate credentials to services acting on their behalf by storing credentials in the MyProxy repository and sending the MyProxy passphrase to the service. They can also use MyProxy to renew their credentials, via the myproxy-get-delegation command, further delegation insures that the lifetime of the new proxy is less than the original to enforce greater security and so, for example, long-running jobs don't fail because of expired credentials. By using a proxy credential delegation protocol, MyProxy allows users to obtain proxy credentials when needed without ever transferring private keys over the network. For users that don't already have PKI credentials, the MyProxy Certificate Authority (CA) provides a convenient method for

obtaining them. The MyProxy CA issues short-lived session credentials to authenticated users. The repository and CA functionality can be combined in one service or can be used separately.

2.2.4.1 MyProxy Certificates useful commands

- **myproxy-init**, uploads a credential to a myproxy-server for later retrieval. In the default mode, the command first prompts for the user's Grid pass phrase, which is used to create a proxy credential. The command then prompts for a MyProxy pass phrase, which will be required to later retrieve the credential. The MyProxy pass phrase must be entered a second time for confirmation. A credential with a predefined lifetime (of one week by default) is then delegated to the myproxy-server and stored with the given MyProxy pass phrase.

- **myproxy-info**, displays information about a user's credentials stored on a myproxy-server. The user must have a valid proxy credential as generated by grid-proxy-init or retrieved by myproxy-get-delegation when running this command.

- **myproxy-get-delegation**, retrieves a credential from the myproxy-server that was previously stored using myproxy-init. In the default mode, the command prompts for the MyProxy pass phrase associated with the credential to be retrieved.

- **myproxy-destroy**, removes the myproxy credentials from the myproxy-server.

For options and other useful commands see Appendix A.2.

3 User level middleware

3.1 Definition

Any Grid infrastructure consists of three basic building blocks: on the side one, *the underlying infrastructure* (or fabric) providing computing and storage resources; on the other side, *the users* with *their applications*, wanting to use the resources. During the first steps of the grid, a number of common issues arose over and over, requiring solutions in order for the application to succeed. These problems suggested the need for design patterns that could be reused from project to project. These design patterns were in a new space between the system level and the application level, that is the mid-level software, hence the term "middleware"; it typically consists of a stack of different modules, which act collectively as an intermediary, hiding from the user the multiple parts and detailed workings of the Grid infrastructure. The Grid thus appears as a single, coherent, easy-to-use resource, in which the middleware ensures that the resources are used as efficiently as possible and in a secure and accountable manner.
Grid pioneers found that it was:

1. too hard to keep track of authentication data (ID/password) across institutions;
2. too hard to monitor system and application status across institutions;
3. too easy to leave "dangling" resources lying around, reducing system robustness.
4. too many ways to submit jobs;
5. too many ways to store, access, and ultimately manage distributed data.

A number of grid initiatives like the Globus Project (now the Globus Alliance), the Global Grid Forum, and the GRIDS Center were founded largely to identify these common challenges, find generic solutions to them, and test the solutions in genuine application settings. Government funding programs like the DOE SciDAC program (http://www.scidac.org/), the NSF Middleware Initiative (NMI - http://www.nsf-middleware.org/default.aspx), the Enabling Grids for E-Science in Europe (EGEE - http://public.eu-egee.org/) initiative, and the UK e-Science Programme (http://www.rcuk.ac.uk/escience/) have invested considerable funding toward this work [http://www.globus.org/]. Grid application projects teach that trying to force homogeneity on distributed groups of collaborators is useless. Each segment of the group will have its own system administrators, its own business office, its own set of rules and regulations that get in the way of homogeneous practices. The collaborators themselves will have their own preferences, which sometimes rises almost to the level of dogma, resisting change or conformity with other ideas. Forcing people to give up their local ways of doing things requires more energy than most projects have at their disposal.
There is a model of how to get a diverse set of incompatible technologies to work together: the Internet. Over the course of several decades, the standard Internet transport protocol (IP) emerged as a mean of sending messages over any kind of physical network. Internet software was developed to allow IP to be run over Ethernet, token ring, wireless and satellite networks, and diverse point-to-point links. Applications could count on IP being available to them regardless of the myriad physical network types being used, and it became significantly easier to develop applications that used network capabilities. The availability and accessibility of the core capability of network message transport led to an explosion in the number of applications that could be developed, including the highly popular World Wide Web, which has become a new "platform" on which new standards can be developed.
Open Grid Services Architecture extends the IP network services upward from physical transport of messages to a variety of services that have proven common in Grid applications. OGSA defines a service-oriented architecture, which can be thought as the key to effective virtualization of physical resources. OGSA services enable applications to address common Grid requirements for on-demand

availability, system management, collaborative computing, and so on. The OGSA builds on existing Web service standards and extends these standards when needed.
The Grid community and the Web services communities initially worked without much coordination, though Grid implementations often used existing Web technologies such as HTTP or public key based security. More recently, the two communities identified a set of common requirements, which has led to the formulation of a set of specifications known as the WS-Resource Framework, or WSRF. WSRF specifications define how to provide Grid capabilities using Web Services implementations. The definition of WSRF means that the Grid and Web services communities can move forward on a common base [http://www.globus.org/grid_software/ecology.php].

3.2 gLite

The gLite middleware is envisaged as a modular system, to allow users to tailor the system to their specific needs by deploying the services they require, rather than using the whole system. The Grid services of gLite follow a Service-Oriented Architecture (SOA), making it easier to connect the software to other Grid services. It will also help users to comply with upcoming Grid standards, for instance the Web Service Resource Framework (WSRF) from OASIS and the Global Grid Forum's Open Grid Service Architecture (OGSA).
Distributed under an open-source license, gLite is mainly based on middleware from EGEE's predecessor, the European DataGrid (EDG) project (http://eu-datagrid.web.cern.ch/eu-datagrid/default.htm), and makes use of components from other ongoing middleware projects. Using existing middleware components developed by many different projects worldwide means that gLite inherits code from other sources. The middleware team in EGEE re-engineers and integrates the different components, which are often written in many different programming languages, into a coherent software suite. Each line of code is measured by how well it complies with coding guidelines and standards defined for each language. Individual units or functions are tested to make sure that they do what they are supposed to do. Functional testing is performed on a distributed test-bed at CERN (http://gridcafe.web.cern.ch/gridcafe/), the CCLRC Rutherford Appleton Laboratory (http://www.cclrc.ac.uk/), NIKHEF (http://www.nikhef.nl/) and Imperial College in London (http://www.lesc.imperial.ac.uk/index.html).
gLite is being developed in two main cycles with incremental releases in between. On 4 April 2005 gLite 1.0 was released to be certified by the operations team, which gathers experience in running middleware on a large-scale production-level service. After repeating the functional tests on an independent test-bed, the middleware is subjected to stress tests. During this certification step, the middleware is also tested on different "flavors" of operating systems used in the EGEE Grid resource centres. Currently this is mainly Scientific Linux, but increasingly other architectures are being requested.
Once the middleware is certified, it is deployed on a preproduction test-bed, which includes 12 sites in Europe and the US that reflect the different sizes and usage policies of sites in the wider EGEE infrastructure. gLite is then exposed to some of EGEE's pilot applications, to test the middleware under real conditions.
The FTS (File Transfer Service) formed part of the incremental release gLite 1.2, which also included a new version of the virtual-organization membership system (VOMS) and further Condor support of the computing elements. gLite 2.0, released at the end of the 2005, focus on secure services, increased robustness and ease of installation, as well as further developments of the core services, incorporating experience gained in the data and service challenges [BER05].

4 EGEE project

4.1 Introduction

The Enabling Grids for E-sciencE (EGEE) project [http://public.eu-egee.org/] brings together scientists and engineers from more than 90 institutions in 32 countries world-wide to provide a seamless Grid infrastructure for e-Science that is available to scientists 24 hours-a-day. Conceived from the start as a four-year project, funded by the European Commission, whose first part officially ended on the 31 March 2006 and whose second two-year phase started on 1 April 2006. The project aimed to provide researchers in academia and industry with access to major computing resources, independent of their geographic location. The EGEE project also focused on attracting a wide range of new users to the Grid.
The project primarily concentrated on some core areas:

1. to build a consistent, robust and secure Grid network that will attract additional computing resources;
2. to continuously improve and maintain the middleware in order to deliver a reliable service to users;
3. to attract new users from industry as well as science and ensure they receive the high standard of training and support they need.
4. to combine national, regional and thematic Grid efforts, for a seamless Grid infrastructure for scientific research and to build a sustainable Grid for business research and industry.

The EGEE Grid was built on the EU Research Network GÉANT and exploited Grid expertise generated by many EU, national and international Grid projects. Funded by the European Commission, the EGEE project community has been divided into 12 partner federation, consisting of over 70 contractors and over 30 non-contracting participants covering a wide-range of both scientific and industrial applications.
The work being carried out in the project was organised into 11 activities which come under three main areas:

- Networking Activities (NA) are the management and coordination of all the communication aspects of the project;
- Service Activities (SA) are the support, operation and management of the Grid as well as the provision of network resources;
- Joint Research Activities (JRA) concentrate on Grid research and development;

Two pilot application domains were selected to guide the implementation and certify the performance and functionality of the evolving infrastructure. One is the Large Hadron Collider Computing Grid (http://lcg.web.cern.ch/LCG/) supporting physics experiments and the other is Biomedical Grids (http://www.healthgrid.org/), where several communities are facing equally daunting challenges to cope with the flood of bioinformatics and healthcare data. Expanding from those originally two scientific fields, high energy physics and life sciences, EGEE now integrates applications from many other scientific fields, ranging from geology to computational chemistry. Generally, the EGEE Grid infrastructure is ideal for any scientific research especially where the time and resources needed for running the applications are considered impractical when using traditional IT infrastructures.
The EGEE Grid consists of over 20,000 CPU available (Figure 6) to users 24 hours a day, 7 days a week, in addition to about 5 Petabytes (5 million Gigabytes) of storage, and maintains 20,000 concurrent jobs on average. Having such resources available changes the way scientific research takes place. The end use depends on the users' needs: large storage capacity, the bandwidth that the

infrastructure provides, or the sheer computing power available. With funding of over 30 million Euros from the European Commission, the project is one of the largest of its kind.

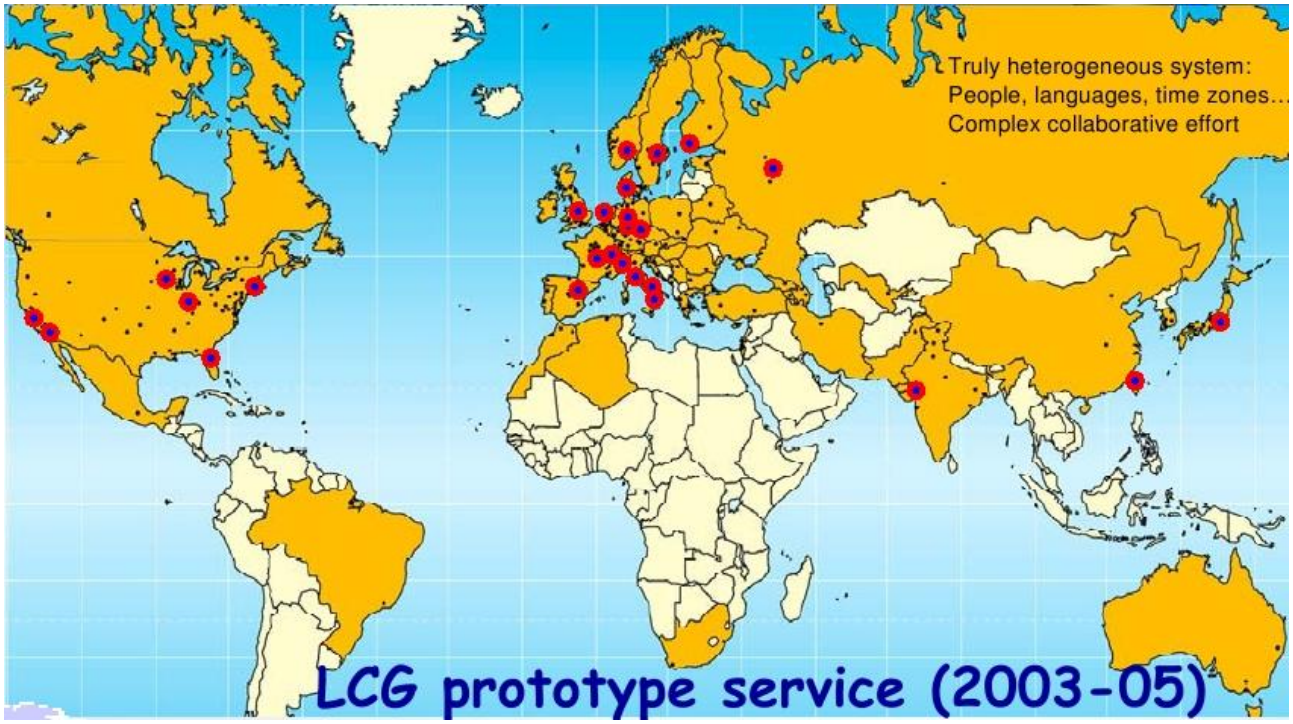


Figure 6: Centres taking part in LCG prototype service (03-05), born to allow CERN scientist to deploy their calculation.

4.2 EGEE Middleware components

The overall structure of the LCG middleware is composed by Workload Management System (WMS), Data Management System (DMS), Information System (IS), Authorization and Authentication System, Accounting System, several monitoring services and installation (Figure 7).
The purpose of the WMS is to accept requests for job submission and management coming from its clients and take the appropriate actions to satisfy them. The complexity of the management of applications and resources in the grid is hidden to the users. The interaction of user with the WMS is indeed limited to the description of the characteristics and requirements for the job via JDL and to the submission of it through the appropriate UI. Then the WMS will translate these abstract resource requirements into a set of actual resources, taken from the overall grid resource pool, to which the user has access permission.
The User Interface (UI) is the component that allows users to access the functionalities offered by the WMS.
The Network Server (NS) is a generic network daemon, responsible for accepting incoming requests from the UI (e.g. job submission, job removal), which, if valid, are then passed to the Workload Manager. When a job is submitted to the WMS, the NS checks if there is enough space to store its input sandbox files.
The Workload Manager (WM) is the core component of the WMS. For a computation job there are two main types of request: submission and cancellation. In particular the meaning of the submission request is to pass the responsibility of the job to the WM. The WM will then pass the job to an appropriate CE for execution, taking into account the requirements and the preferences expressed in the job description. The decision of which resource should be used is the outcome of a matchmaking process between the submission requests and the available resources. The availability of resources for a particular task depends not only on the state of the resources, but also on the

utilisation policies that the resource administrators and/or the administrator of the VO the user belongs to have put in place. Given a valid request, it has to take the appropriate actions to satisfy it. To do so, it may need support from other components, which are specific to the different request types. All these components that offer support to the WM provide a class whose interface is inherited from a Helper class. Essentially the Helper, given a JDL expression, returns a modified one, which represents the output of the required action. For example, if the request was to find a suitable resource for a job, the input JDL expression will be the one specified by the user, and the output will be the JDL expression augmented with the CE choice.
The Resource Broker (RB) or Matchmaker is one of these classes offering support to the WM. It provides a matchmaking service: given a JDL expression (e.g. for a job submission), it finds the resources that best match the request. It interacts with the Information Service and with the data management services.
A WM can adopt different policies in order to schedule a job.

- a job is matched to a resource as soon as possible and, once the decision has been taken, the job is passed to the selected resource for execution.
- the jobs are held by the WM until a resource becomes available (eager scheduling), at which point that resource is matched against the submitted jobs and the job that fits best is passed to the resource for immediate execution (lazy scheduling policy).

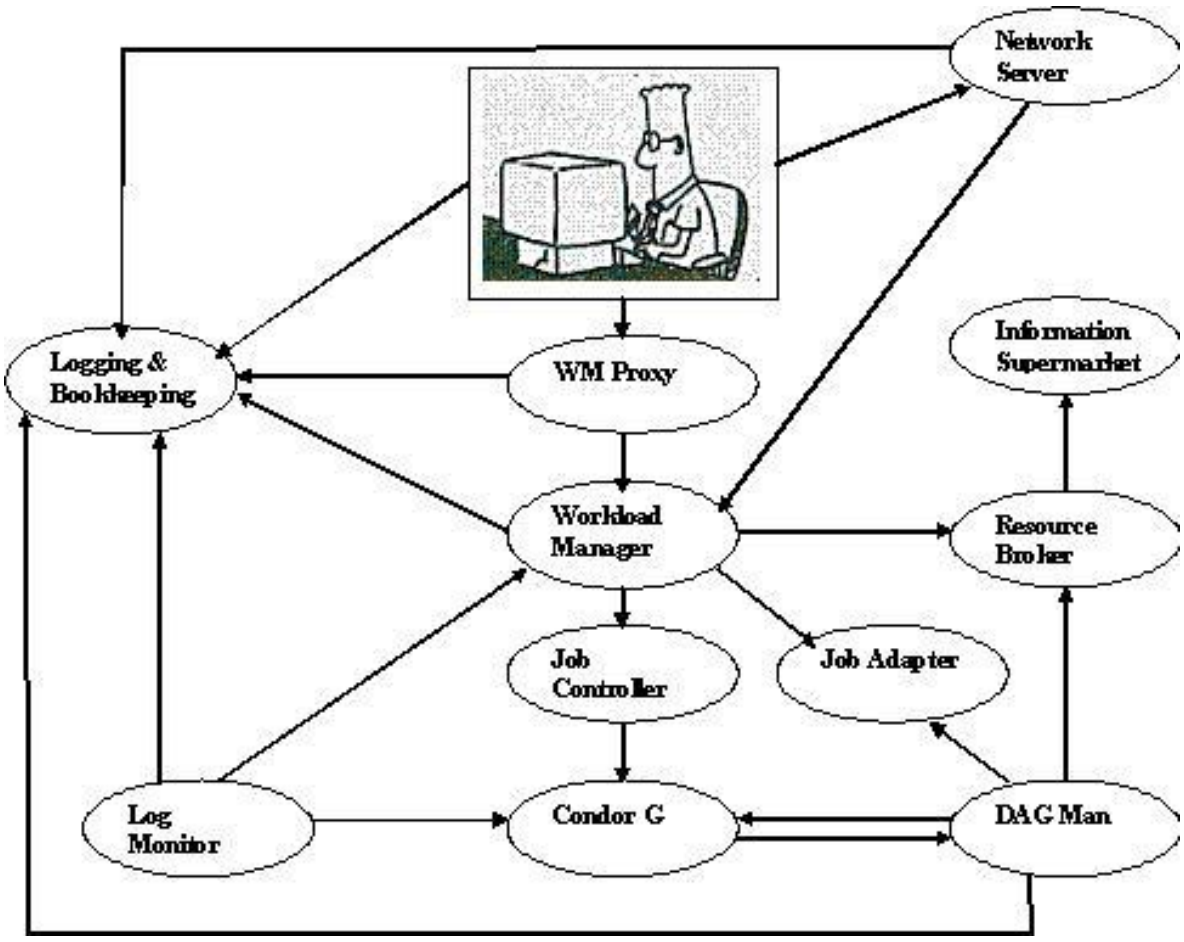


Figure 7: schema of EGEE middleware components.

The mechanism that allows the flexible application of different policies is the decoupling between the collection of information concerning resources and its use. The Information Super Market (ISM) is the component that implements this mechanism and basically consists of a repository of resource information that is available in read only mode to the matchmaking engine and whose update is the

result of either the arrival of notifications or active polling of resources or some arbitrary combination of both.
The other fundamental component of the WM internal design is the Task Queue (TQ), that gives the possibility to keep a submission request for a while if no resources are immediately available that match the job requirements.
This technique is used, among others, by the AliEn and Condor systems. Non matching requests will be retried either periodically (in an eager scheduling approach) or as soon as notifications of available resources appear in the ISM (in a lazy scheduling approach). Alternatively such situations could only lead to an immediate abort of the job for lack of a matching resource.
Continuing with the WMS components handling the job during its lifetime, we have the Job Adapter (JA) which is responsible for making the final "touches" to the JDL expression for a job, before it is passed to CondorG for the actual submission. So, besides preparing the CondorG submission file, this module is also responsible for creating the wrapper script, and for creating the appropriate execution environment in the CE worker node (this includes the transfer of the input and of the output sandboxes).
CondorG is the module responsible for performing the actual job management operations (job submission, job removal, etc.), issued on request of the Workload Manager.
DAGMan (DAG Manager) is a meta-scheduler whose main purpose is to navigate the graph (i.e. the DAG request), determine which nodes are free of dependencies, and follow the execution of the corresponding jobs. A DAGMan instance is spawned by CondorG for each handled DAG.
The Log Monitor (LM) is responsible for "watching" the CondorG log file, intercepting interesting events concerning active jobs, that is events affecting the job state machine (e.g. job done, job cancelled, etc.), and therefore triggering appropriate actions.
For what concerns the Logging and Bookkeeping (LB) service, it stores logging and bookkeeping information concerning events generated by the various components of the WMS. Using this information, the LB service keeps a state machine view of each job.

4.3 GILDA INFN – Test bed

EGEE project builds Grids, specifically a very large scale production Grid for scientists around the world. The development of this project's major product actually requires the construction of many smaller Grids for a variety of tasks such as software testing and pre-production. It also requires resources to aid demonstrations and in-depth training sessions for users and administrators.
While simulations and other virtual training systems might have been used, EGEE has access to a 'real world' training system: the GILDA (the Grid INFN virtual Laboratory for Dissemination Activities) testbed [https://gilda.ct.infn.it/testbed.html].
Launched in 2004 by the EGEE partner INFN (Istituto Nazionale di Fisica Nucleare), GILDA is a fully working Grid testbed devoted to dissemination activities, and allows both users and system administrators access to first hand experience with Grid systems. In the context of EGEE, GILDA acts as a crucial component of the project's t-Infrastructure (t stands for training) programme, helping to pass on knowledge and experience, as well as computing resources, to the scientific community and industry.
The GILDA testbed consists of 20 sites on three continents (Figure 8), using heterogeneous hardware to act as a 'real world' grid environment. It is made up of all the components of a larger Grid project, including testing and monitoring systems. To allow use of the testbed by any interested party, it also features a Virtual Organisation and a real Certification Authority (CA) that grants two-week certificates for the test use of the GILDA infrastructure. To tailor GILDA to the t-Infrastructure needs of EGEE, it features a number of different portals for different uses, including basic portals for first time users and full featured ones for more in depth tutorials and demonstrations.

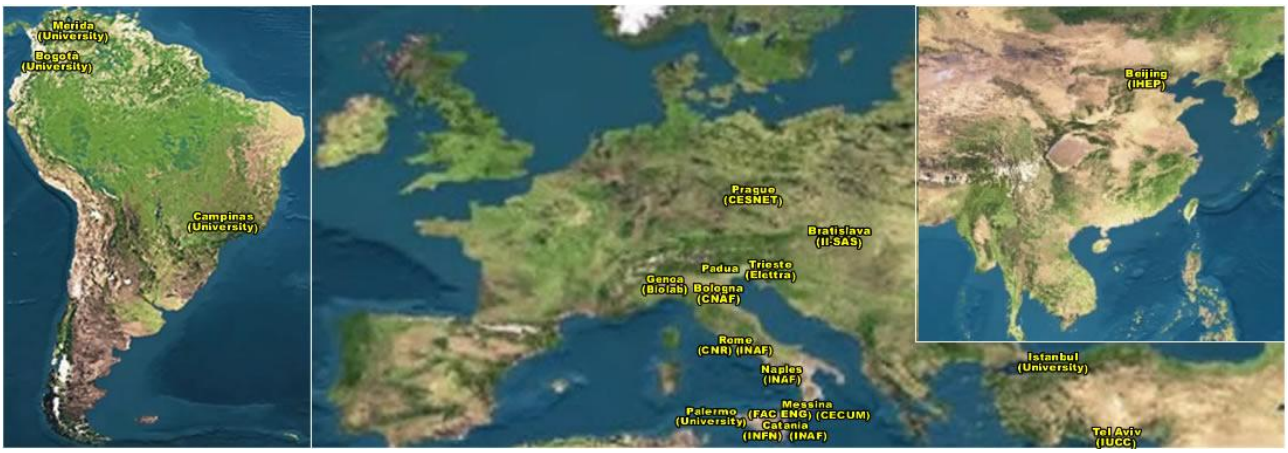


Figure 8: GILDA nodes sites

GILDA fits seamlessly into the EGEE training and dissemination activities on a number of levels. It acts as a first point of contact for new users, allowing them quick and easy access to a working Grid and experience with the system of Virtual Organisations and CA used in Grids across the world. Later, it also acts as a resource for EGEE training activities such as demonstrations, tutorials and hands-on sessions. So far it has been used for more than 100 induction courses and demonstrations, with its website generating in excess of 1,200,000 hits. Finally, GILDA acts as a miniproduction environment, allowing users to test applications ported to the EGEE infrastructure. The GILDA testbed is a very useful resource in its own right, but also forms part of the 'Virtuous Cycle', the process whereby EGEE aims to recruit new user communities. As well as giving first hand grid experience, successful use of the GILDA testbed aids to recruitment of future users by providing an increasing pool of applications for use in Grid demonstrations. GILDA is open to anyone who wants to try out the Grid, be they EGEE partners or members of the public. By some simple steps, users are able to get quick and easy access to the Grid, as a demonstration but also for the full Grid experience (including obtaining an authentication certificate and joining a Virtual Organisation). In addition, GILDA runs gLite, EGEE's new lightweight middleware solution, keeping in step with the project and allowing users access to cutting edge technology. At a more advanced level, GILDA offers a one-of-a-kind service for those interested in testing the Grid, gLite and the EGEE infrastructure with their own systems.

4.4 Actual Job submission workflow

A certificate is required for each user to prove his/her identity when he/she submits a request to the testbed. Resource Brokers, Storage Elements and Replica Catalogs also need certificates to prove their identity. A testbed is made up of one or more sites. Each site contains a certain number of machines, each one playing a different role. Each role is implemented by one or more middleware modules that have to be installed on the machine. Below is a list of the different roles:

- The User Interface is the module that allows users to access all the DataGrid service (Job submission, Data management, Information management, etc.);
- The Resource Broker is the module that receives users requests and queries the Information Index to find suitable resources;
- The Information Index, which can reside on the same machine as the Resource Broker, keeps information about the available resources;
- The Storage Element is the module installed on the machines which will provide storage space to the testbed. It provides a uniform interface to different Storage Systems;

- The Replica Manager is used to coordinate file replication across the testbed from one Storage Element to another. This is useful for data redundancy but also to move data closer to the machines which will perform computation;
- The Replica Catalog, which can reside on the same machine as the Replica Manager, keeps information about file replicas. A logical file can be associated to one or more physical files which are replicas of the same data. Thus a logical file name can refer to one or more physical file names;
- The Computing Element is the module which receives job requests and delivers them to the Worker Nodes, which will perform the real work. The Computing Element provides an interface to local batch queuing systems (e.g. PBS, LSF, ...). A Computing Element manages one or more Worker Nodes. A Worker Node can also be installed on the same machine as the Computing Element;
- The Worker Node is the module installed on the machines which will process input data;

4.5 Commands

Main actions when submitting a work are submission, check list of free hardware, check of work status, work deletion, output retrieval. To each one of those actions corresponds a middleware command.

4.5.1 Job Submission: glite-job-submission

Is the command for submitting jobs to the Grid; it allows the user to run a job at one or several remote resources. glite-job-submit requires as input a job description file in which job characteristics and requirements are expressed. While it does not matter the order of the other arguments, the job description file has to be the last argument of this command. After successful submissions, this command returns the identifier (JobId) assigned to the job.
It is possible to specify some options, see Appendix C.1

4.5.2 Check Status: glite-job-status

This command prints the status of a job previously submitted using glite-job-submit. The job status request is sent to the LB that provides the requested information. This can be done during the whole job life. glite-job-status can monitor one or more jobs: the jobs to be checked are identified by one or more jobIds (returned by glite-job-submit) provided as arguments to the command and separated by a blank space. It is possible to specify some options, see Appendix C.2

4.5.3 Job Deletion: glite-job-cancel

This command cancels a job previously submitted using glite-job-submit. Before cancellation, it prompts the user for confirmation. The cancel request is sent to the Network Server that forwards it to the WM that fulfils it. glite-job-cancel can remove one or more jobs: the jobs to be removed are identified by their jobIds (returned by glite-job-submit) provided as arguments to the command and separated by a blank space. The result of the cancel operation is reported to the user for each specified jobId. It is possible to specify some options, see Appendix C.3.

4.5.4 Getting the Output: glite-job-output

The glite-job-output command can be used to retrieve the output files of a job that has been submitted through the glite-job-submit command with a job description file including the

OutputSandbox attribute. After the submission, when the job has terminated its execution, the user can download the files generated by the job and temporarily stored on the RB machine as specified by the OutputSandbox attribute, issuing the glite-job-output with as input the jobId returned by the glite-job-submit. It is possible to specify some options, see Appendix C.4.

4.5.5 Checking list of available hardware: glite-job-list-match

This command displays the list of identifiers of the resources, satisfying the job requirements included in the job description file, on which the user is authorized. The CE identifiers are returned either on the standard output or in a file according to the chosen command options, and are strings univocally identifying the CEs published in the IS. The returned CEIds are listed in decreasing order of rank, i.e. the one with the best (greater) rank is in the first place and so forth.
It is possible to specify some options, see Appendix C.5.

4.5.6 Getting logging info: glite-job-logging-info

This command queries the LB persistent DB for logging information about jobs previously submitted using glite-job-submit. The job logging information are stored permanently by the LB service and can be retrieved also after the job has terminated its life-cycle, differently from the bookkeeping information that are in some way "consumed" by the user during the job existence.
It is possible to specify some options, see Appendix C.6.

5 Scripting

5.1 Introduction

A principal consideration of resource management systems is the efficient assignment of resources to customers. The problem of making such efficient assignments is referred to as the resource allocation problem and it is commonly formulated in the context of a scheduling model that includes a system model. A system model is an abstraction of the underlying resources, to describe the availability, performance characteristics and allocation policies of the resources being managed.
The system model provides information to the allocator regarding the availability and properties of resources at any point in time [RAM98].
Distributed ownership together with heterogeneity, resource failure and evolution make it impossible to formulate a monolithic system model, there is therefore a need for a resource management paradigm that does not require such a model and that can operate in an environment where resource owners and customers dynamically define their own models. Matchmaking uses a semi-structured data model - the classified advertisements data model - to represent the principals of the system and folds the query language into the data model, allowing entities to publish queries (i.e., requirements) as attributes [SOL04].

A fundamental notion for workload management in any such distributed and heterogeneous environment is entities (i.e. servers and customers) description, which is accomplished with the use of a description language.

5.2 ClassAd Language

The Classified Advertisement (ClassAd) language is a symmetric description language (both servers and customers use the same language to describe their respective characteristics, constraints and preferences) whose central construct is the classad, a record-like structure composed of a finite number of distinct attribute names mapped to expressions. A classad is a highly flexible and extensible data model that can be used to represent arbitrary services and constraints on their allocation [JDL01]. The classad language is a simple expression-based language that enables the specification of many interesting and useful resource and customer policies facilitating the operation of identification and ranking of compatible matches between resources and customers. It has as its major goal to allow the easy matching between resources and requests, to correctly have jobs executed on the Grid. The Classad language is therefore the language "spoken" by the WMS components that are directly involved in the job submission process i.e. UI, RB and JSS.

It is based on simple descriptional statements in the general format:

```
attribute = value;
```

where values can be of different types: numeric, strings, booleans, timestamps etc.

Some of these attributes are used to describe the technical characteristics of the job to be submitted, some other attributes are used to specify requirements for a computing element which is supposed to be found by the RB and to be the executor of the given Job, specifying a given set or constraints and preferences on the executor node to be found.

The classad language differentiates between expressions and values: expressions are evaluable language constructs obtained by parsing valid expression syntax, whereas values are the results of evaluating expressions.

5.3 JDL Attributes

The JDL is a fully extensible language, hence it is allowed to use whatever attribute for the description of a job. Anyway only a certain set of attributes that we will refer as "supported attributes" from now on, is taken into account by the Workload Management System components in order to schedule a submitted job.
The supported attributes can be grouped into two main categories:

- Resources attributes
- Job attributes

Resource attributes are those that have to be used to build expressions of the Requirements and Rank attributes in the job class-ad and to be effective, i.e. to be actually used for selecting a resource, have to belong to the set of characteristics of the resources that are published in the GIS (aka MDS).
Job attributes represent instead job specific information and specify in some way actions that have to be performed by the RB to schedule the job. Some of these attributes are provided by the user when he/she edits the job description file while some others (needed by the RB) are inserted by the UI before submitting the job.
A small subset of the attributes that are inserted by the user is mandatory, i.e. necessary for the RB to work correctly and can be split in two categories:

- Mandatory: the lack of these attributes does not allow the submission of the job
- Mandatory with default value: the UI provides default value for these attributes if they are missing in the job description [PAC05].

5.3.1 Main Attributes

Attribute	M	With default	Meaning
Type		Job	• **Job** a simple job • **DAG** a Direct Acyclic Graph of dependent jobs • **Collection** a set of independent jobs
Executable	✔		Executable/command name. The user can specify an executable that lies already on the remote CE. The absolute path, possibly including environment variables, of this file should be specified. The other possibility is to provide a local executable name, which will be staged on the CE. In this case only the file name has to be specified as executable. The absolute path on the local file system executable should be then listed in the *InputSandbox* attribute expression
RetryCount	✔	3	Number of job submission retrial made by JSS in case the submission fails for some reason. Default value is 3.
ReplicaCatalog	✔ **(*)**		Replica Catalogue Identifier, i.e. something in the following format: <protocol>://<full hostname> :<port>/<Replica Catalog DN>. **(*)** This attribute is **mandatory if** the *InputData*

Attribute	M	With default	Meaning
			attribute has been also specified.
Rank	✔ (**)	-Estimated TraversalTime	a ClassAd Floating-Point expression that states how to rank queues that have already met the Constraints expression. Essentially, rank expresses preference. A higher numeric value equals better rank. The RB will give the job the queue with the highest rank. Default value for this attribute is: -*EstimatedTraversalTime* (**) This value is always added to the Rank expression; if the user has provided a value then the rank is: Rank= < expression > - *EstimatedTraversalTime*
Requirements	✔	TRUE	Boolean ClassAd expression which uses C-like operators. It represents job requirements on resources. In order for a job to run on a given queue, this Constraint expression must evaluate to true on the given queue. Default value for this attribute is TRUE.

For other attributes see Appendice B.

6 Application

6.1 Introduction to CODESA-3D

CODESA-3D [GAM99] is a three-dimensional finite element simulator for flow and solute transport in variably saturated porous media on unstructured domains. The flow and solute transport processes are coupled through the variable density of the filtrating mixture made of water and dissolved matter. The flow module simulates the water movement in the porous medium, taking into account different forcing inputs, while the transport module computes the migration of the salty plume due to advection and diffusion processes [LEC00].
CODESA-3d is born from the coupling of two parent codes developed in 1990s at University of Padova and CRS4:

- SATC3D for saturated flow and transport;
- FLOW3D for unsaturated flow;

Some details on CODESA mathematical model can be found on Appendice E

6.2 Aims of our program

The aim of this work is to develop an application to run a Montecarlo application of CODESA-3D software in two different hardware infrastructures, a cluster of PC and the EGEE production Grid.
The analysis consists of results comparison, of execution time comparaison and run effectiveness.
This paper should be seen not as an end-point work, but instead as a start-point work for studying the effectiveness of the GRID architecture when running medium-sized codes.
Moreover should be helpful as a guide-line during the development of the CyberSar project (2006-2008, financed by MIUR http://www.cybersar.com).

6.3 JDL script

As seen before the first step preparing a job submission on a GRID is to write a *file.jdl*, where are located job characteristics and user requirements.

6.3.1 COD3d.jdl

The file COD3d.jdl is a simple jdl file with the following characteristics.

Is a Normal executable job; the type attribute is mandatory.

Type = "Job";
JobType = "Normal";

The execution of *"start_root.sh"* is done using *sh* program in the */bin/* directory. This condition specifies that this execution can run just on nodes where the program *sh* has path */bin/sh*.

Executable = "/bin/sh" ;
Arguments = "start_root.sh" ;

Those files are useful to read any execution message and/or error message generated during the execution.

StdOutput = "std.out";
StdError = "std.err";

To not overcome the quota transfer that is 2 Mb, in this case we create a .tar archive of all files needed to run the job, and gzip them.

InputSandbox = {"start_root.sh",
"./codesa-muravera.i686.gz",
"./FEFLOW/input.tar.gz",
"./FEFLOW/nansfneubc.s.gz"};

In the output sandbox we want to have the output produced from the file, again compressed and gzipped to not exceed the quota transfer.

OutputSandbox = {"std.err","output.tar.gz"};

If the submission fails for any reason, the jdl try a new submission.

RetryCount=2;

6.3.2 start_root.sh

The executable, first gunzips and explodes the tar.gz files then changes the modes of main executable "codesa-muravera.i686", then executes CODESA-3D program.
After the end of the execution, the output files are gzipped and a .tar archive is created.

```
gunzip input.tar.gz
tar -xf  input.tar
gunzip *.gz
chmod 777 codesa-muravera.i686
# Start execution of the codesa file.
echo "Start execution of CODESA3D.."
./codesa-muravera.i686
echo "..Done!"
gzip *.out *.dat *.OUT
tar -cf output.tar *.out.gz *.dat.gz *.OUT.gz
gzip output.tar
```

6.3.3 CODESA3D_Multirun.sh

To submit the job a shell script named "*CODESA3D_Multirun.sh*" is launched from command-line, while another one that investigates the production status and named "*Polling_Production.sh*" is called from the first one.
First step is to edit "*CODESA3D_Multirun.sh*" file to set some variables connected to the execution, as the working directory, the name of file.jdl needed to submit the job on grid, etc.
This script can submit jobs on glite middleware or on old edg middleware. The choice is made setting the tag "*GLITE_UI*" respectively as "TRUE" or "FALSE"; the default value is *"TRUE"*.
Before the run an archive of directories and subdirectories where the files with all useful information about the job execution will be written, is created.

It is possible to execute "*./CODESA3D_Multirun.sh [option][name][input][number]*" with five different [options]:

- –s

        - To submit a new production, for a more detailed description see Appendix D.1;
    - –l
        - To list the submitted productions, for a more detailed description see Appendix D.1;
    - –m
        - To monitor the status of the productions, for a more detailed description see Appendix D.1;
    - –a
        - To retrieve the productions output, for a more detailed description see Appendix D.1;
    - –c
        - To clean the environment, for a more detailed description see Appendix D.1;
- [name]
    - Is the Production name submitted, is mandatory with –s, –m and –a options;
- [input]
    - is the DataSet used to create "*file.jdl*"
- [number]
    - is the number of production events

If the call to "*CODESA3D_Multirun.sh*" contains any mistake (either wrong number of arguments, or option not allowed, or a production name already used, or non-existent input directory, or not natural number of runs), the instructions to run it correctly will appear on the screen.

6.3.3.1 submission (option –s)

First there is a check on the existence of the data set, on the existence of the file.jdl, on the uniqueness of the production name.
Then are created: the file containing JOB_ID associated to the job and obtained as first output of the submission; the file with status of execution; the directory which will contain all the jobs outputs. The date and time of execution is contained also into some names of directories or files.
Before starting the actual job submission the script tests if there are nodes where the job could be run using the command "*glite-job-list-match {file.jdl}*", already described on 4.5.5.
After those checks the submission starts with the command "*glite-job-submit [options] {file.jdl}*" already described on 4.5.1.

6.3.3.2 listing (option –l)

With this option a list of available productions, is written on the screen

6.3.3.3 status (option –m)

With this option, followed by the name of an available production, the status of execution of the production is printed on the screen.
The status can be STARTED or EXECUTED.

6.3.3.4 output retrieval (option –a)

There is a check on the production status, if the production is EXECUTED, the output is retrieved, with the command "*glite-job-output – [dir output_directory] JOBID* " already described on 4.5.4.

6.3.3.5 cleaning (option –c)

Kill the polling process if it is still active. Remove temporary files and all files connected with the production name if specified.

6.3.4 Polling_Production.sh

This script is called from "*CODESA3D_Multirun.sh*".
Its task is to update the files with the status of single jobs (ready, waiting, submitted, scheduled, running, aborted, done, cleared) of the production during the execution. It updates also the file with the status of the production changing it from STARTED to EXECUTED when each production job is run and is arrived in a status of done or aborted or cancelled; another task is to write some files with useful information as date and time of execution or as JOB_ID of jobs still running. For more details see Appendix D.2

6.4 CRS4 cluster – shell scripting;

A cluster of Personal Computers (PCs) was initiated at CRS4 in October 1999. In the first quarter of 2006, 48 nodes with dual-processor dualcore boards were added, named janas01 - janas48. CPU: AMD Opteron Dualcore Processor 265, 1800 MHz, 1 MB cache, two per cluster node. Four core per nodeMemory: 4 GigaByte, DDR 400 MHz Registered ECC. Two local hard disk SATA for scratch files. Communication: Gigabit Ethernet. Linux operating system, CentOS 4.2, 64-bit addressing. The operating system is Linux. Software from the Portland Group was purchased for the languages Fortran 90, High Performance Fortran (HPF) and OpenMP.

6.4.1 SGEEE 5.3 - Sun Grid Engine Enterprise Edition 5.3

A cluster system can be seen as a grid (*Cluster grids* are the simplest grids, consisting of computer *hosts* working together to provide a single point of access to users in a single project or department) with single owner, single site, single organization.
SGE enables both the users and the computational cluster to get the most out of available resources. Users can treat the collection of compute nodes like a single system, and submit jobs from any of the nodes. SGE will automatically run jobs on less loaded nodes, regardless of which node the jobs were submitted on; SGE will also queue jobs for later execution to avoid overloading available resources and causing the entire system to operate poorly for everyone.
From the user perspective, there are two major components to a batch queue: the node hardware and the job execution queues. The hardware determines the available computational capacity. The queues define the system job-execution policy, *e.g.*, how many jobs can run simultaneously and how much CPU time a job can consume. Typically there are multiple queues, each defining a different policy, and users can choose which queue to use for a particular job.
SGEEE 5.3 software orchestrates the delivery of computational power based upon enterprise resource *policies* set by the organization's technical and management staff. The Sun Grid Engine system uses these policies to examine the available computational resources within the cluster grid, gathers these resources, and then allocates and delivers them automatically in a way that optimizes usage across the cluster grid.
SGEEE system:

- is transparent to the user;
- distributes the workload homogeneously on the available nodes;
- keeps track of done work, and let the user to access to this information;
- has some tools for monitoring and controlling jobs and hosts;

The SGEEE system accepts jobs - users requests for computer resources - from the outside world, puts them in a holding area until they can be executed, sends them from the holding area to an execution device, manages them during execution, and logs the record of their execution when they are finished [SUN02].
Sun Grid Engine (SGE) is an open source batch queuing system. Users submit jobs which are placed in queues and the jobs are then executed, depending on the system load, the opening hours of the queues and the job priorities, when the resource requirements of the job are fulfilled. A queue can be defined as a parallel queue containing a number of slots. When there are no more free slots the job is let pending and waiting for free resources. Resource use policy is "Equal-share-sort" this prevents a user from "pushing" other users downwards by submitting a series of jobs (from a shell-script).
The user can submit batch jobs, interactive jobs, and parallel jobs to the SGEEE, either via command line or via QMON that is a graphic user interface.

6.4.2 Useful instructions for job submission and monitoring

Each submitted job must be a shell script.
Via command line the instruction can be "*qsub [option] job.sh*".
options:
-a CCYYMMDDhhmmss -cwd -M (user@domain) -m base
-e /path/file_stderr -o /path/file_stdout -p priority -l resource=value
With "*qalter [options] jobID*" it is possible to modify the queries related to the job.
With "*qconf*" we can investigate about the status of the queue.
We can ask the system about the status of the job with the instruction "*qstat [option] jobid*". A job can be:

- r - running (execution);
- t - transferring (it is going to be executed);
- d - deletion (deletion from queues);
- s - suspended (by the user);
- S - suspended (by the system);
- T - threshold (suspended by the system for threshold problems);
- w - waiting (insufficient resources);
- h - hold (waiting more appropriate information);

It is possible to delete the jobs with the instruction "*qdel jobnum*".
When a job has ended, the console output of the script will be put into files in the users home directory; if a user submits the job "simple.sh" the system will answer: *your job simple.sh has been submitted*, if there is no different specifications, two files named *script.ojobnum.* (stdout) and *script.ejobnum (stderr)*, will be created.

7 Results

To enhance our knowledge about GRID we did a big amount of trial runs obtaining also as result a better customization of shell scripts used during the submission and the fixing of some bugs.
After this first period, some jobs to try and give a measure of EGEE-GRID performance, were run.

A Monte Carlo simulation composed by 100 single jobs, each one having a different input stochastic parameter value, was run for 100 times; the stochastic value is represented by the pumping rate of the wells.

The interest was in collecting information about run time and successful jobs percentage, to obtain a measure of grid reliability.

During job execution, we were not interested on output post processing so we got just elaboration time.

This procedure demonstrated itself to be wrong, because of an overwhelming of the resource broker storage element, where the output sandbox is stored, waiting for a command of job-get-output that gets the output in the user home directory and frees the disk space.

This wrong approach killed all runs for 56 groups and killed partially 14 of the others.

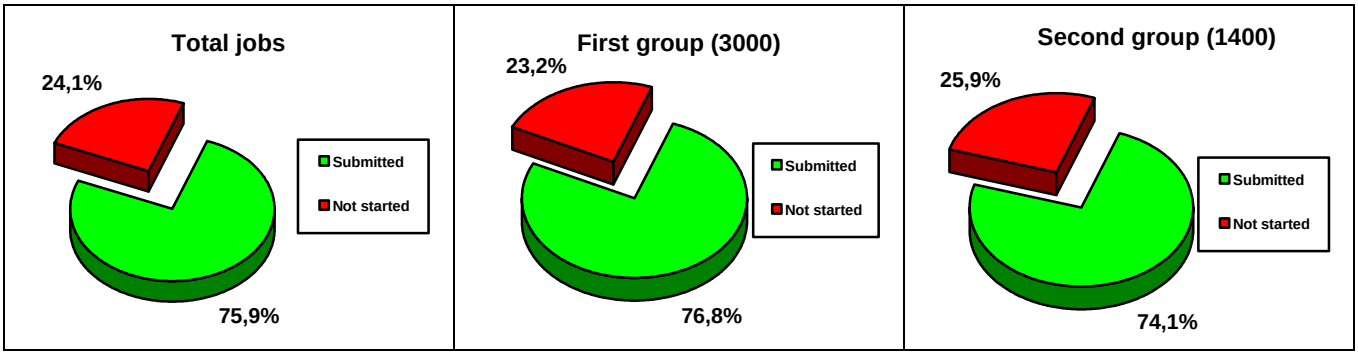
7.1 Numerical considerations

We got those results:

- ~ 100 Groups started
- ~ 56 groups killed
- ~ 14 groups partially killed
- ~ 30 groups ended
    - 14 of ended groups, arrived at the end autonomously
    - 16 of ended groups needed to be killed after 10 hours, because of pending jobs.

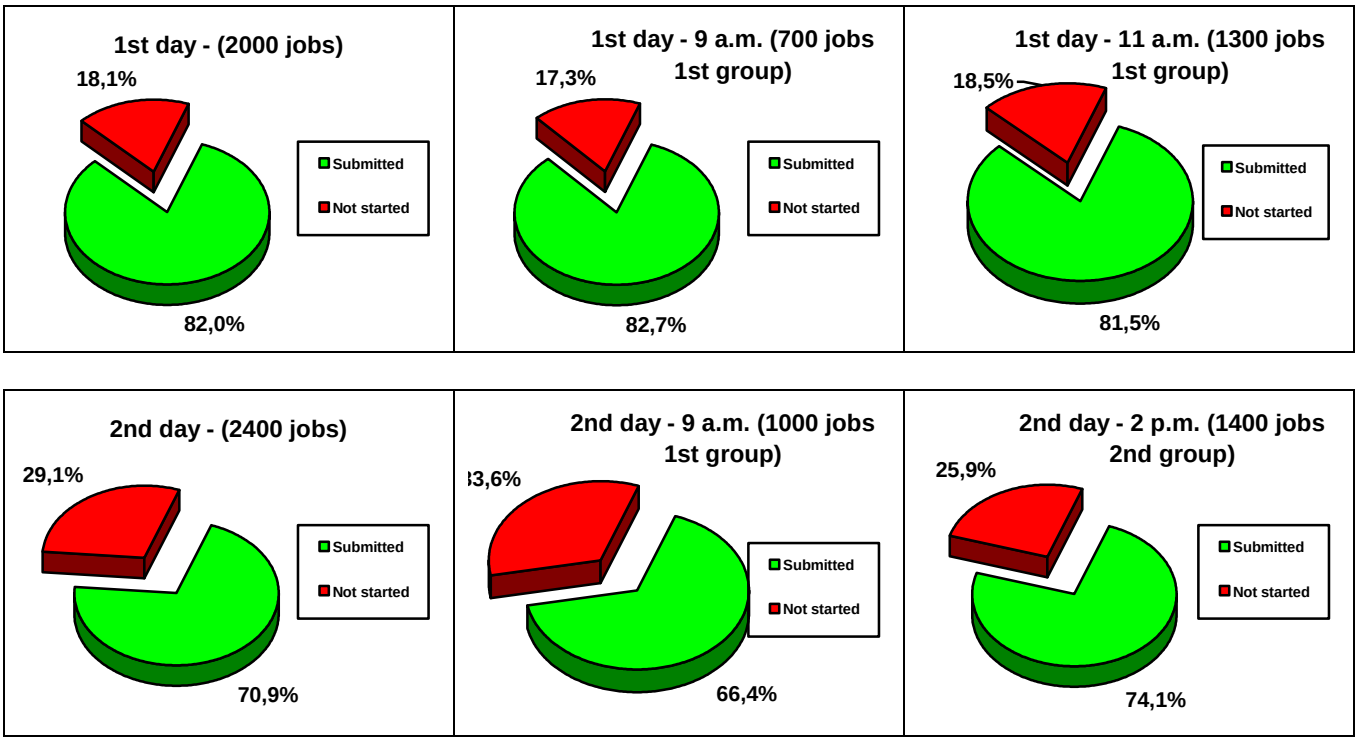
So the analysis was done just on 44 groups. First of all 44 run were grouped all together and in a second step they were divided on 2 smaller subgroups the first of it composed by the 30 groups completed and the second one composed with the lasting 14.

7.1.1 All groups together

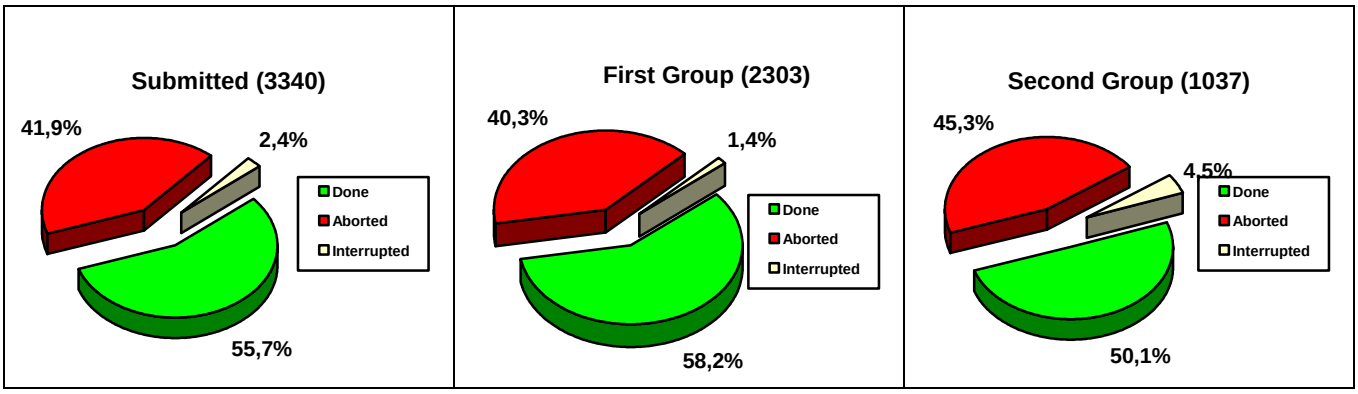


It could be noticed considering 4400 jobs altogether that a percentage almost equal to 25 % of jobs was lost before submission, and that this percentage is almost constant in each subgroup.

It is maybe interesting to do some elaboration based on submission time, because the first group, the one with 3000 jobs, was subgrouped and submitted in two consecutive days in three different times respectively at 9 a.m., at 11 a.m., at 9 a.m. The group with 1400 jobs was submitted entirely the second day at 2 p.m.
The results are drawn on those sequent graphics:

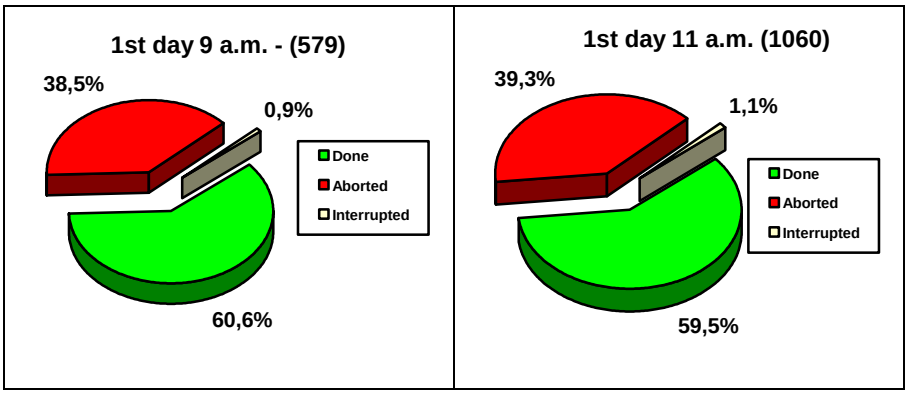


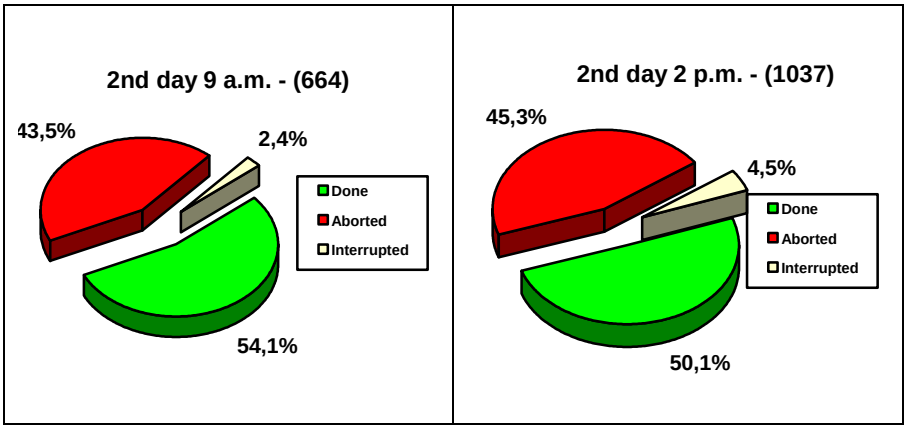
The first day the percentage of submitted jobs is bigger than 80%, the second day shows an inconstant percentage at best lower than 75% instead.
Considering just the submitted jobs, we found that:



Considering the jobs that achieved the end and flagged as “done” at the end of the running we noticed that just about the 56% gave a useful result. About 40 % of the jobs aborted and about 4% of the jobs stayed pending for a lot of time.
There is a difference of 8% between the two groups considered; a percentage lower than 60% of successful jobs can be considered not so useful although it could be improved either running the interrupted groups again and again till the completion of the group or waiting for a longer time before stopping the pending jobs. But the number of undone jobs could indicates that there are some troubles on working nodes.

In the sequent graphics we printed the result of some elaboration done considering submission day and submission time of submitted jobs:

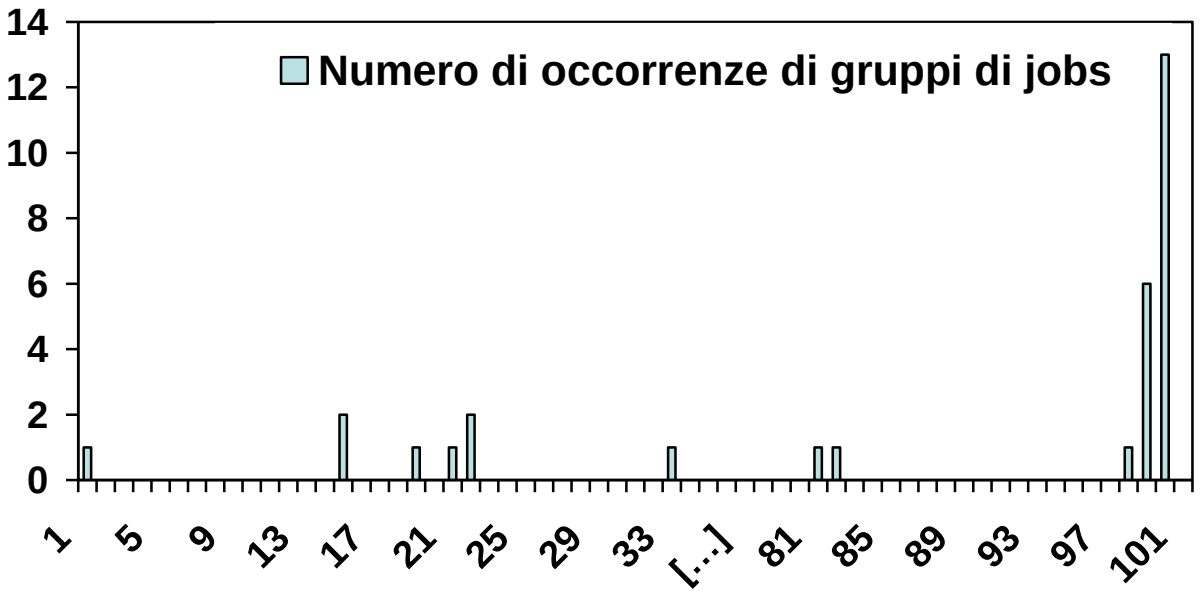




The percentage of jobs done is 60% and almost constant during the first day of submission, and during the second day varies of 4% going from 54% of 1st group to 50% of 2nd group.
The best result of second day is lower than the almost constant result on first day, a possible explanation is a different performance on available working nodes or a different performance on net connection and data transfer.

7.1.2 Submission number

For each group 100 jobs should be submitted, but the actual number of submitted jobs is highly variable. Just 13 times all the 100 jobs were submitted, in other 6 times the jobs submitted were 99, in another one 98.
The other groups that are under 85 jobs submitted, are useless for a Monte Carlo statistics because of their incompleteness.
But each one of the 100 jobs is deterministic, so the groups that got an interruption before 100 jobs submitted could be completed, running the missing jobs, and become part of the statistics.



7.1.3 Submission time

Some other consideration could transpire considering the submission time of the packages.
The scripts record the time from the submission of the first job, till the time of submission of the last job really submitted.
We choice to consider for our purposes just the groups having at least 98 jobs submitted, to make a comparison between similar packages.
Submission time varied between just 28' 06" and 2^h 18' 13". The minimum is very different and distant from the other values, that are about 2^h 5' 0".
Two global means were calculated the first one considering all the values between the minimum and the maximum is 2^h 0' 22"; the second one was calculated excluding the values extremely different form the others and is 2^h 5' 47".
We calculated also the mean values of the subgroups composed by clustering together the groups submitted at almost the same time and we obtained 3 mean values:

- 1^{st} day – 9 a.m. $\rightarrow$ 1^h 52' 36"
- 1^{st} day – 11 a.m. $\rightarrow$ 2^h 17' 14"
- 2^{nd} day – 9 a.m. $\rightarrow$ 1^h 59' 24"

7.1.4 Completed groups and Execution time

Execution time means the time from the first job submission till the execution of the last job pending.
Inside the script there is an instruction to interrupt the jobs pending after about 10 hours, with the command "glite-job-cancel" so those jobs are not considered in the statistics.
We spoke of completed groups when each job submitted achieved an autonomous conclusion during its run on the GRID.
Just 13 groups completed their run, the minimum execution time was 2^h 30' 41" and the maximum was 6^h 36' 48".
The mean time was 3^h 31' 22".

Subgrouping the completed groups according their submission time, we obtain:

- 1^{st} day – 9 a.m. $\rightarrow$ 2^{h} 31' 42"
- 1^{st} day – 11 a.m. $\rightarrow$ 5^{h} 29' 47"
- 2^{nd} day – 9 a.m. $\rightarrow$ 2^{h} 32' 39"

7.2 Conclusions

The number of jobs considered (4400) is small, but this could be the starting point for a work of study about GRID performances.

Such a statistics is interesting for users because the GRID does not mean just augmented computing power, but also reliability.

It is difficult to give any definitive conclusion or opinion about it, but those numbers could furnish some hints and reminders to proceed the study of GRID performance.

The percentage of job done is very low, considering the submission time, running again the jobs undone to have complete results means that GRID advantages are lost.

We did not know why the jobs aborted. There is a consistent number of pending jobs that persist to be pending for a great amount of time; if the user needs all the outputs to do some post processing elaboration, pending jobs must be considered as lost jobs.

The runs on CRS4 cluster gave better results than those on EGEE GRID production; group submission on batch modality took a couple of minutes; although the execution time on cluster varies according to the number of other jobs on queue, usually it took just a few minutes to execute the group of 100 jobs.

Our scripts were not optimized neither considering the data transfer on grid, nor considering the possibility to use MPI and parallel computing. Anyway it seems that the performance could depend on the day and time of submission, sometimes GRID runs faster and someday it runs slower, the hour of submission seems to be connected with the submission day.

Acknowledgments. The author thanks Giuseppe La Rocca and Roberto Barbera of the INFN-GRID group of Catania (Italy), and Philippe Renard and Jaouher Kerrou of the University of Neuchatel (Switzerland).

Bibliography

[PAR05] M. Parashar, C. A. Lee. *Scanning the Issue: Special Issue on Grid-Computing***.**
Proceedings of the IEEE, 93(3):479--484, March 2005.

[BUY05] R. Buyya, S. Venugopal. *A Gentle Introduction to Grid Computing and Technologies*.
CSI Communication VOL 9:9--19, July 2005

[COR65] F. J. Corbat, V. A. Vyssotsky. *Introduction and overview of the multics system*.
Proc. Amer. Fed. Information Processing Soc. 1965 Fall Joint Computer Conf., vol. 27: 185–196.

[WAL02] D. W. Walker. *Emerging Distributed Computing Technologies*.
CMP13 Department of Computer Science - Cardiff University

[FOS03] I. Foster and C. Kesselman. *The Grid, Blueprint for a New Computing Infrastructure*.
2nd ed. Morgan Kaufmann, 2003

[FOS01] I. Foster, C. Kesselman, S. Tuecke *The Anatomy of the Grid Enabling Scalable Virtual Organizations*. International J. Supercomputer Applications, 15(3), 2001.

[AIK00] R. Aiken, M. Carey, B. Carpenter, I. Foster, C. Lynch, J. Mambretti, R. Moore, J. Strasnner, B. Teitelbaum. *Network Policy and Services: A Report of a Workshop on Middleware*,
IETF, RFC 2768, 2000.

[CAT92] C. Catlett, L. Smarr. *Metacomputing*.
Commun. ACM, vol. 36, no. 6: 44–52, 1992.

[BER02] F. Berman, G. Fox and T. Hey. *The Grid: Past, Present and Future*
From the book: *Grid Computing: Making the Global Infrastructure a Reality*. 2002 John Wiley & Sons, Ltd

[FOS02] I. Foster, C. Kesselman, J. M. Nick, S. Tuecke. *The Physiology of the Grid: An Open Grid Services Architecture (OGSA) for Distributed Systems Integration***.**
Open Grid Service Infrastructure WG, Global Grid Forum, June 22, 2002.

[JAC06] B. Jacob. *Grid computing: What are the key components?*
IBM DeveloperWorks, White Paper, updated 2006.
http://www-28.ibm.com/developerworks/grid/library/gr-overview/

[FER03] L. Ferreira, V. Berstis, J. Armstrong, M. Kendzierski, A. Neukoetter, M. Takagi, R. Bing-Wo, A. Amir, R. Murakawa, O. Hernandez, J. Magowan, N. Bieberstein. *Introduction to Grid Computing with Globus*.
IBM redbook - International Technical Support Organization - 2003

[GEL04] M. Geldof. *The Semantic Grid: will Semantic Web and Grid go hand in hand?*
European Commission DG Information Society Unit “Grid technologies” June 2004

[TUE02] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, T. Maguire, T. Sandholm, P. Vanderbilt, D. Snelling. *Open Grid Services Infrastructure (OGSI) version 1.0.*
Technical report, Global Grid Forum, 2002.

[PAT03] P. Pattnaik, K. Ekanadham, J. Jann. *Autonomic Computing and Grid*.
From the book: *Grid Computing*, Pages 351-361. 2003 John Wiley & Sons, Ltd

[ZHU04] H. Zhuge. *The Knowledge Grid*
World Scientific Publishing Company. December 2004

[JOH02] W. E. Johnston. *Computational and data Grids in large-scale science and engineering*. Future Generation Computer Systems, Volume 18 , Issue 8:1085-1100 (October 2002)

[ALL01] W. Allcock, A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, S. Tuecke. *The data grid: Towards an architecture for the distributed management and analysis of large scientific datasets*.
Journal of Network and Computer Applications, 23:187-200, 2001.

[CAS02] H. Casanova. *Distributed Computing Research Issues in Grid Computing*.
ACM SIGAct News, Vol. 33, No. 3, 2002, pp. 50 – 70.

[NAD05] K. Nadiminti, R. Buyya. *Enterprise Grid computing: State-of-the-Art*.
Enterprise Open Source Journal, Dallas, Texas, USA, March/April 2006.

[FOS05] I. Foster. *Globus Toolkit Version 4:Software for Service-Oriented Systems*.
IFIP International Conference on Network and Parallel Computing, Springer-Verlag LNCS 3779, pp 2-13, 2005.

[FOS97] I. Foster and C. Kesselman. *Globus: A Metacomputing Infrastructure Toolkit*. International Journal of Supercomputer Applications, 11(2):115--128, 1997.

[BER05] R. Berlich, M. Kunze, K. Schwarz. *Grid computing in Europe: from research to* ***deployment.***
Proc. of the 2005 Australasian workshop on Grid computing and e-research - Volume 44: 21 - 27

[DIF88] W. Diffie. *The first ten years of public-key cryptography*.
Proceedings of the IEEE 76 (1988), 560-577.

[NOV01] J. Novotny, S. Tuecke, V. Welch. *An Online Credential Repository for the Grid: MyProxy*.

Proc. of the Tenth International Symposium on High Performance Distributed Computing (HPDC-10), IEEE Press, August 2001, pages 104-111.
[GAM99] G. Gambolati, M. Putti, C. Paniconi. *Three-dimensional model of coupled density-dependent flow and miscible salt transport in groundwater* – from the book: Bear et al. Eds, *Seawater Intrusion in Coastal Aquifers: Concepts, Methods, and Practices, chapter 10*.
Kluwer Academic, Dordrecht, Holland.
[LEC00] G. Lecca. *Implementation and testing of the CODESA-3D model for density dependent flow and transport problems in porous media*.
CRS4 –Technical Report – 00/40, CRS4, Cagliari, Italy, 2000.
[RAM98] R. Raman, M. Livny, M. Solomon. *Matchmaking: Distributed Resource Management for High Throughput Computing*.
Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing, July 28-31, 1998, Chicago, IL.
[SOL04] M. Solomon. *The ClassAd Language Reference Manual Version 2.4*.
Computer Sciences Department University of Wisconsin—Madison. May, 2004
[JDL01] Datamat SpA. *Data Grid - Job Description Language – How to*. 2001
[PAC05] F. Pacini. *JDL attributes specification*.
EGEE Document EGEE-JRA1-TEX-555796-JDL-Attributes-v0-1, 3 Feb. 2005.
[SUN02] AA. VV., *Sun™ ONE Grid Engine, Enterprise Edition Administration and User's Guide*.
Sun Microsystems Manuals, October 2002

Useful Links:

EUMEDGRID: http://www.eumedgrid.org/
DEGREE: http://www.eu-degree.eu/DEGREE
EGEE 1: http://public.eu-egee.org/
EGEE 2: http://www.eu-egee.org/
GILDA: https://gilda.ct.infn.it/
GRID café from CERN: http://gridcafe.web.cern.ch/gridcafe/
GLOBUS: http://www.globus.org/
INFN GRID: http://www.infn.it/indexen.php

Appendice A. Proxy and MyProxy attributes

A.1 Proxy certificate useful commands

A.1.1 **glite-voms-proxy-init** (grid-proxy-init, edg-proxy-init, etc.) Some options are:

- **-help, -usage**
  - Displays usage
- **-version**
  - Displays version
- **-debug**
  - Enables extra debug output
- **-quiet, -q**
  - Quiet mode, minimal output
- **-verify**
  - Verifies certificate to make proxy for
- **-pwstdin**
  - Allows passphrase from stdin
- **-limited**
  - Creates a limited proxy
- **-valid <h:m>**
  - Proxy is valid for h hours and m minutes (default to 12:00)
- **-hours H**
  - Proxy is valid for H hours (default:12)
- **-bits**
  - Number of bits in key {512|1024|2048|4096}
- **-cert <certfile>**
  - Non-standard location of user certificate
- **-key <keyfile>**
  - Non-standard location of user key
- **-certdir <certdir>**
  - Non-standard location of trusted cert dir
- **-out <proxyfile>**
  - Non-standard location of new proxy cert
- **-voms <voms<:command>>**
  - Specify voms server. :command is optional.
- **-order <group<:role>>**
  - Specify ordering of attributes.
- **-target <hostname>**
  - Targets the AC against a specific hostname.
- **-vomslife <h:m>**
  - Try to get a VOMS pseudocert valid for h hours and m minutes (default to value of -valid).
- **-include <file>**
  - Include the contents of the specified file.
- **-conf <file>**
  - Read options from <file>.
- **-confile <file>**
  - Non-standard location of voms server addresses.
- **-userconf <file>**
  - Non-standard location of user-defined voms server addresses.
- **-vomses <file>**
  - Non-standard location of configuration files.
- **-policy <policyfile>**
  - File containing policy to store in the ProxyCertInfo extension.
- **-pl, -policy-language <oid>**
  - OID string for the policy language.
- **-policy-language <oid>**
  - OID string for the policy language.
- **-path-length <l>**

        - Allows a chain of at most l proxies to be generated from this one.
    - **-globus**
        - Globus version.
    - **-proxyver**
        - Version of proxy certificate.
    - **-noregen**
        - Doesn't regenerate a new proxy for the connection.
    - **-separate <file>**
        - Saves the information returned by the server on file <file>.
    - **-ignorewarn**
        - Ignore warnings.
    - **-failonwarn**
        - Treat warnings as errors.
    - **-list**
        - Show all available attributes.

A.1.2 **glite-voms-proxy-info** (grid-proxy-info, edg-proxy-info, etc.). Some options are:

- **-help, -usage**
    - Displays usage
- **-version**
    - Displays version
- **-debug**
    - Displays debugging output
- **-file <proxyfile>**
    - Non-standard location of proxy
- **[printoptions]**
    - Prints information about proxy and attribute certificate
    - **-subject**
        - Distinguished name (DN) of proxy subject
    - **-issuer**
        - DN of proxy issuer (certificate signer)
    - **-identity**
        - DN of the identity represented by the proxy
    - **-type**
        - Type of proxy (full or limited)
    - **-timeleft**
        - Time (in seconds) until proxy expires
    - **-strength**
        - Key size (in bits)
    - **-all**
        - All proxy options in a human readable format
    - **-text**
        - All of the certificate
    - **-path**
        - Pathname of proxy file
    - **-vo**
        - Vo name
    - **-fqan**
        - Attribute in FQAN format -acsubject Distinguished name (DN) of AC subject
    - **-acissuer**
        - DN of AC issuer (certificate signer)
    - **-actimeleft**
        - Time (in seconds) until AC expires
    - **-serial**
        - AC serial number
- **-exists** [options]
    - Returns 0 if valid proxy exists, 1 otherwise
    - [options to -exists] (if none are given, H = B = 0 are assumed)
        - **-valid H:M**

        - time requirement for proxy to be valid
    - **-hours H**
        - time requirement for proxy to be valid (deprecated, use -valid instead)
    - **-bits B**
        - strength requirement for proxy to be valid
- **-acexists** <voname>
    - Returns 0 if AC exists corresponding to voname, 1 otherwise

A.1.3 **glite-voms-proxy-destroy** (grid-proxy-destroy, edg-proxy-destroy, etc.). Some options are:

- **-help, -usage**
    - Displays usage
- **-version**
    - Displays version
- **-debug**
    - Enabless extra debug output
- **-file <proxyfile>**
    - Specifies proxy file name
- **-dry**
    - Only go in dryrun mode
- **-quiet, -q**
    - Quiet mode, minimal output
- **-conf <file>**
    - Read options from <file>.

A.2 My proxy server useful commands

A.2.1 myproxy-init

- **-h, --help**
    - Displays command usage text and exits.
- **-u, --usage**
    - Displays command usage text and exits.
- **-v, --verbose**
    - Enables verbose debugging output to the terminal.
- **-V, --version**
    - Displays version information and exits.
- **-s** *hostname*, **--pshost** *hostname*
    - Specifies the hostname of the myproxy-server. This option is required if the MYPROXY_SERVER environment variable is not defined. If specified, this option overrides the MYPROXY_SERVER environment variable.
- **-p** *port*, **--psport** *port*
    - Specifies the TCP port number of the myproxy-server. Default: 7512.
- **-P, --pidfile** *path*
    - Specifies a file to write the pid to.
- **-l, --username**
    - Specifies the MyProxy account under which the credential should be stored. by default, the command uses the value of the LOGNAME environment variable. Use this option to specify a different account username on the MyProxy server. The MyProxy username need not correspond to a real Unix username.
- **-c** *hours*, **--cred_lifetime** *hours*
    - Specifies the lifetime of the credential stored on the myproxy-server in hours. Specify *0* for the maximum possible lifetime, i.e., the lifetime of the original credential. Default: 1 week (168 hours).
- **-t** *hours*, **--proxy_lifetime** hours
    - Specifies the maximum lifetime of credentials retrieved from the myproxy-server using the stored credential. Default: 12 hours.
- **-d, --dn_as_username**
    - Use the *certificate subject* (DN) as the default username, instead of the LOGNAME environment variable.
- **-a, --allow_anonymous_retrievers**

        - Allow credentials to be retrieved with just pass phrase authentication. By default, only entities with credentials that match the myproxy-server.config default retriever policy may retrieve credentials. This option allows entities without existing credentials to retrieve a credential using pass phrase authentication by including "anonymous" in the set of allowed retrievers. The myproxy-server.config server-wide policy must also allow "anonymous" clients for this option to have an effect.
    - **-A, --allow_anonymous_renewers**
        - Allow credentials to be renewed by any client. Any client with a valid credential with a subject name that matches the stored credential may retrieve a new credential from the MyProxy repository if this option is given. Since this effectively defeats the purpose of proxy credential lifetimes, it is not recommended. It is included only for the sake of completeness.
    - **-r** *dn*, **--retrievable_by** *dn*
        - Allow the specified entity to retrieve credentials. By default, the argument will be matched against the common name (CN) of the client (for example: "Fabrizio Murgia"). Specify **-x** before this option to match against the full distinguished name (DN) (for example: "/C=EU/O=CRS4/CN=Fabrizio Murgia") instead.
    - **-R** *dn*, **--renewable_by** *dn*
        - Allow the specified entity to renew credentials. By default, the argument will be matched against the common name (CN) of the client (for example: "condorg/modi4.ncsa.uiuc.edu"). Specify **-x** before this option to match against the full distinguished name (DN) (for example: "/C=US/O=National Computational Science Alliance/CN=condorg/modi4.ncsa.uiuc.edu") instead. This option implies **-n** since passphrase authentication is not used for credential renewal.
    - **-x, --regex_dn_match**
        - Specifies that the DN used by options **-r** and **-R** will be matched as a regular expression.
    - **-X, --match_cn_only**
        - Specifies that the DN used by options **-r** and **-R** will be matched against the Common Name (CN) of the subject.
    - **-k** *name*, **--credname** *name*
        - Specifies the credential name.
    - **-K** *description*
    - **--creddesc** *description*
        - Specifies credential description.
    - **-S, --stdin_pass**
        - by default, the command prompts for a passphrase and reads the passphrase from the active tty. When running the command non-interactively, there may be no associated tty. Specifying this option tells the command to read passphrases from standard input without prompts or confirmation.

A.2.2 myproxy-info

- **-all**
    - print all the information about the my-proxy credential stored on the myproxy-server
- **-s** *hostname*, **--pshost** *hostname*
    - Specifies the hostname of the myproxy-server. This option is required if the MYPROXY_SERVER environment variable is not defined. If specified, this option overrides the MYPROXY_SERVER environment variable.
- **-p** *port*, **--psport** *port*
    - Specifies the TCP port number of the myproxy-server. Default: 7512.

A.2.3 myproxy-get-delegation

- **-t** *hours*, **--proxy_lifetime** *hours*
    - Specifies the lifetime of credentials retrieved from the **myproxy-server** using the stored credential. The resulting lifetime is the shorter of the requested lifetime and the lifetime specified when the credential was stored using **myproxy-init**. Default: 12 hours.

A.2.4 myproxy-destroy

- removes the myproxy credentials from the **myproxy-server**.

A.2.5 myproxy-store

- Store end-entity credential for later retrieval

A.2.6 myproxy-retrieve

- Retrieve an end-entity credential

A.2.7 myproxy-change-pass-phrase

- Change a credential's passphrase

A.2.8 myproxy-admin-adduser

- Add a new user credential

A.2.9 myproxy-admin-change-pass

- Change credential passphrase

A.2.10 myproxy-admin-query

- Query repository contents

A.2.11 myproxy-admin-load-credential

- Directly load repository

A.2.12 myproxy-server

- Store credentials in an online repository

Appendice B. JDL Attributes

B.1 Main Attributes

Attribute	M	With default	Meaning
Type		Job	• **Job** a simple job • **DAG** a Direct Acyclic Graph of dependent jobs • **Collection** a set of independent jobs
Job type		Normal	• **Normal** a simple batch job • **Interactive** a job whose standard streams are forwarded to the submitting client • **MPICH** a parallel application using MPICH-P4 implementation of MPI • **Partitionable** a job that can be thought as composed by a set of independent steps/iterations, i.e. a set of independent sub-jobs, each one taking care of a step or of a sub-set of steps, and which can be executed in parallel • **Checkpointable** a job able to save its state, so that the job execution can be suspended, and resumed later, starting from the same point where it was first stopped • **Parametric** a job whose JDL contains parametric attributes (e.g. Arguments, StdInput etc.) whose values can be made vary in order to obtain submission of several instances of similar jobs only differing for the value of the parameterized attributes
Executable	✔		Executable/command name. The user can specify an executable that lies already on the remote CE. The absolute path, possibly including environment variables, of this file should be specified. The other possibility is to provide a local executable name, which will be staged on the CE. In this case only the file name has to be specified as executable. The absolute path on the local file system executable should be then listed in the *InputSandbox* attribute expression
InputData			A single or a list of: - logical collections and/or - logical files and/or - physical files This attribute refers to data used as input by the job; these data are stored in SEs and published in replica catalogues. Wildcards are admitted in the specification of this attribute.
StdInput			Standard input of the job. It can be: - just a file name (staging required) - absolute path (available on the CE) The same mechanism as described for the *Executable* attribute can be applied.
StdOutput			Standard output of the job. The user has to specify just the file name. To have this file staged back on the submitting machine he/she has to list the file name also in the *OutputSandbox* attribute expression and use the **dg-get-job-output** command.
StdError			Standard error of the job. The user has to specify just the file name. To have this file staged back on the submitting machine he/she has to list the file name also in the *OutputSandbox* attribute expression and use the **dg-get-job-output** command.

Attribute	M	With default	Meaning
OutputSE			URI of the Storage Element where to store the output data. Once specified, this attribute shall be used to build the job requirements expression in order make the RB choose a resource being "attached" with this SE. For example: *Requirements = ... && Member(OutputSE, other.CloseSE);*
InputSandbox			List of files on the UI local disk needed by the job for running. The listed files are staged from the UI to the remote CE. Wildcards are admitted in the specification of this attribute. This attribute is also used to accomplish executable and standard output staging from the submitting machine to the remote execution CE.
OutputSandbox			List of files, generated by the job, which have to be retrieved. The listed files are transferred on the UI local file system by mean of the **dg-get-job-output** command. Wildcards are admitted in the specification of this attribute.
RetryCount	✔	3	Number of job submission retrial made by JSS in case the submission fails for some reason. Default value is 3.
ReplicaCatalog	✔ **(*)**		Replica Catalogue Identifier, i.e. something in the following format: <protocol>://<full hostname> :<port>/<Replica Catalog DN>. **(*)** This attribute is **mandatory if** the *InputData* attribute has been also specified.
Rank	✔ **(**)**	-Estimated TraversalTime	a ClassAd Floating-Point expression that states how to rank queues that have already met the Constraints expression. Essentially, rank expresses preference. A higher numeric value equals better rank. The RB will give the job the queue with the highest rank. Default value for this attribute is: -*EstimatedTraversalTime* **(**)** This value is always added to the Rank expression; if the user has provided a value then the rank is: Rank= < expression > - *EstimatedTraversalTime*
Requirements	✔	TRUE	Boolean ClassAd expression which uses C-like operators. It represents job requirements on resources. In order for a job to run on a given queue, this Constraint expression must evaluate to true on the given queue. Default value for this attribute is TRUE.

B.2 Job attributes by UI

Attribute	Meaning
dg_jobId	Grid-wide unique job identifier assigned by the UI to the job before submission. Format of the ob identifier is *<LBname>/<UIname>/<time><PID><RND>/<RBname>* where: - *LBname* is the LB server name and port - *UIname* is the UI machine hostname - *time* is the current time on the submitting machine in *hhmmss* format - *PID* is the UI process identifier (if more UI instances are running on the same machine) - *RND* is a random number generated at each job submission - *RBname* is the RB hostname and port
CertificateSubject	Subject of the X509 user certificate. The user's certificate is searched in the file indicated by X509_USER_CERT environment variable. If the variable is not set the default is: ~/.globus/usercert.pem This attributes is matched by the RB with the list of users authorized to

Attribute	Meaning
dg_jobId	Grid-wide unique job identifier assigned by the UI to the job before submission. Format of the ob identifier is *<LBname>/<UIname>/<time><PID><RND>/<RBname>* where: - *LBname* is the LB server name and port - *UIname* is the UI machine hostname - *time* is the current time on the submitting machine in *hhmmss* format - *PID* is the UI process identifier (if more UI instances are running on the same machine) - *RND* is a random number generated at each job submission - *RBname* is the RB hostname and port
	submit job to the CE, represented by the *AuthorizedUser* resource attribute published in the GIS.
UserContact	This is a valid e-mail address where the job status changes notifications have to be sent. This attribute is set by the UI when the user issues the **dg-job-submit** command with *–notify* option.
ResourceId	Grid-wide unique identifier of a resource published in the GIS. This attribute is set by the UI when the user issues the **dg-job-submit** command with *–resource* option and makes the RB directly submit the job to specified resource.

B.3 Resource attributes

Attribute	Meaning
ResourceManagementType	Defines the type of resource management system (LSF/Condor/…).
ResourceManagementVersion	The version of the local resource management system.
GRAMVersion	the GRAM version.
Architecture	the architecture of the machine or of the machines associated to the queue (we assume that all the machines "belonging" to the queue have the same architecture).
OpSys	the operating system of the machine or of the machines associated to the queue (assuming that all these machines run the same operating system).
PhysicalMemory	Minimum available physical memory (in bytes) associated to the queue.
LocalDisk	Local disk footprint
TotalCPUs	the number of total CPUs associated to the resource.
FreeCPUs	the total number of free processors associated to the resource, processors able to run, in that moment, jobs submitted to the resource.
NumSMPs	number of SMP processors associated to the resource.
TotalJobs	the number of jobs submitted to the resource, jobs that have not already been completed.
RunningJobs	The number of jobs submitted to the resources that are currently running.
IdleJobs	the number of jobs submitted to the resource, jobs that are not running since they are waiting for available resources.
MaxTotalJobs	the maximum number of jobs (running and idle) allowed for the resource.
MaxRunningJobs	the maximum number of running jobs allowed for the resource.

Attribute	Meaning
WorstTraversalTime	Worst traversal time
EstimatedTraversalTime	Scaled value of the last traversal time, i.e. *(Last job traversal time)*(queue length) /(queue length when that job arrived)*
Status	the status of the resource. For a queue if it is ready or not to dispatch jobs to the executing machines.
RunWindows	the time windows that define when the resource is active, (for a queue: ready to dispatch jobs to the executing machines). This attribute may appear zero or more times for a Computing Element entity, i.e. it has a list value-type.
Priority	the priority of the resource.
MaximumCPUTime	the maximum CPU time allowed for jobs submitted to the resource.
MaximumWallClockTime	the maximum wall clock time allowed for jobs submitted to the resource.
MinSI00	It is the minimum value of the SpecInt2000 benchmark among the processors associated to this CE. If the CE is a "single processor", this value represents its actual performance.
MaxSI00	It is the maximum value of the SpecInt2000 benchmark among the processors associated to this CE. If the CE is a "single processor", this value represents its actual performance.
AvgSI00	It is the average of the SpecInt2000 benchmark of the nodes associated to this CE. If the CE is a "single processor", this value represents its actual performance.
AuthorizedUser	This is the subject of an X509 user certificate, representing a user authorized to submit job to the CE. This attribute may appear zero or more times for a ComputingElement entity (value-type is list)
RunTimeEnvironments	It is a tag defining a run time environment/package/software installed on the Computing Element. In case, the version of this package/environment is included in this string. This attribute may appear zero or more times for a ComputingElement entity (value-type is list).
AFSAvailable	Boolean attribute defining if AFS is installed on the Computing Element.
OutboundIP	Boolean. It indicates if outbound connectivity is allowed (e.g. all the worker nodes associated to the CE can "initiate" a data transfer, sending and/or receiving data to/from a remote Internet node).
InboundIP	Boolean. It indicates if inbound connectivity is allowed (e.g. a remote Internet node can "initiate" a data transfer, sending and/or receive data to/from any worker node associated to the CE).
CloseSE	Is the string that univocally identifies a Storage Element close enough to the computing element. This attribute may appear zero or more times for a CE entity, i.e. it can be a list of values.

Appendice C. Submission Commands

C.1 glite-job-submission

- **--version**
  - displays UI version.
- **--help**
  - displays command usage
- **--config**, **-c** < *configfile* >
  - if the command is launched with this option, the configuration file pointed by configfile is used. This option is meaningless when used together with "--vo" option
- **--debug**
  - When this option is specified, debugging information is displayed on the standard output and written into the log file, whose location is eventually printed on screen. The default UI logfile location is: glite-wms-job-<command_name>_<uid>_<pid>_<time>.log located under the /var/tmp directory please notice that this path can be overriden with the '--logfile' option
- **--logfile** < *filepath* >
  - when this option is specified, all information is written into the specified file pointed by filepath. This option will override the default location of the logfile: glite-wms-job-<command_name>_<uid>_<pid>_<time>.log located under the /var/tmp directory
- **--noint**
  - if this option is specified, every interactive question to the user is skipped and the operation is continued (when possible)
- **--input**, **-i** < *filepath* >
  - if this option is specified, the user will be asked to choose a CEId from a list of CEs contained in the filepath. Once a CEId has been selected the command behaves as explained for the resource option. If this option is used together with the --int one and the input file contains more than one CEId, then the first CEId in the list is taken into account for submitting the job.
- **--output**, **-o** < *filepath* >
  - writes the generated jobId assigned to the submitted job in the file specified by filepath, which can be either a simple name or an absolute path (on the submitting machine). In the former case the file is created in the current working directory.
- **--resource**, **-r** < *ceid* >
  - This command is available only for jobs. if this option is specified, the job-ad sent to the NS contains a line of the type "SubmitTo = <ceid>" and the job is submitted by the WMS to the resource identified by <ceid> without going through the match-making process.
- **--nodes-resource** < *ceid* >
  - This command is available only for dags. if this option is specified, the job-ad sent to the NS contains a line of the type "SubmitTo = <ceid>" and the dag is submitted by the WMS to the resource identified by <ceid> without going through the match-making process for each of its nodes.
- **--nolisten**
  - This option can be used only for interactive jobs. It makes the command forward the job standard streams coming from the WN to named pipes on the client machine whose names are returned to the user together with the OS id of the listener process. This allows the user to interact with the job through her/his own tools. It is important to note that when this option is specified, the command has no more control over the launched listener process that has hence to be killed by the user (through the returned process id) once the job is finished.
- **--nogui**
  - This option can be used only for interactive jobs. As the command for such jobs opens an X window, the user should make sure that an X server is running on the local machine and if she/he is connected to the UI node from a remote machine (e.g. with ssh) enable secure X11 tunnelling. If this is not possible, the user can specify the --nogui option that makes the command provide a simple standard non-graphical interaction with the running job.
- **--nomsg**
  - this option makes the command print on the standard output only the jobId generated for the job when submission was successful. The location of the log file containing massages and diagnostics is printed otherwise.
- **--chkpt** < *filepath* >
  - This option can be used only for checkpointable jobs. The state specified as input is a checkpoint state generated by a previously submitted job. This option makes the submitted job start running from the

checkpoint state given in input and not from the very beginning. The initial checkpoint states to be used with this option can be retrieved by means of the glite-job-get-chkpt command.

- **--lrms** < *lrmstype* >
  - This option is only for MPICH jobs and must be used together with either --resource or --input option; it specifies the type of the lrms of the resource the user is submitting to. When the batch system type of the specified CE resource given is not known, the lrms must be provided while submitting. For non-MPICH jobs this option will be ignored.
- **--valid**, **-v** < *hh:mm* >
  - A job for which no compatible CEs have been found during the matchmaking phase is hold in the WMS Task Queue for a certain time so that it can be subjected again to matchmaking from time to time until a compatible CE is found. The JDL ExpiryTime attribute is an integer representing the date and time (in seconds since epoch)until the job request has to be considered valid by the WMS. This option allows specifying the validity in hours and minutes from submission time of the submitted JDL. When this option is used the command sets the value for the ExpiryTime attribute converting appropriately the relative timestamp provided as input. It overrides, if present, the current value. If the specified value exceeds one day from job submission then it is not taken into account by the WMS.
- **--config-vo** < *configfile* >
  - if the command is launched with this option, the VO-specific configuration file pointed by configfile is used. This option is meaningless when used together with "--vo" option
- **--vo** < *voname* >
  - this option allows the user to specify the name of the Virtual Organisation she/he is currently working for. If the user proxy contains VOMS extensions then the VO specified through this option is overridden by the default VO contained in the proxy (i.e. this option is only useful when working with non-VOMS proxies). This option is meaningless when used together with "--config-vo" option

C.2 glite-job-status

- **--version**
  - displays UI version.
- **--help**
  - displays command usage
- **--config**, **-c** < *configfile* >
  - if the command is launched with this option, the configuration file pointed by configfile is used. This option is meaningless when used together with "--vo" option
- **--debug**
  - When this option is specified, debugging information is displayed on the standard output and written into the log file, whose location is eventually printed on screen. The default UI logfile location is: glite-wms-job-<command_name>_<uid>_<pid>_<time>.log located under the /var/tmp directory please notice that this path can be overriden with the '--logfile' option
- **--logfile** < *filepath* >
  - when this option is specified, all information is written into the specified file pointed by filepath. This option will override the default location of the logfile: glite-wms-job-<command_name>_<uid>_<pid>_<time>.log located under the /var/tmp directory
- **--noint**
  - if this option is specified, every interactive question to the user is skipped and the operation is continued (when possible)
- **--input**, **-i** < *filepath* >
  - Allow the user to select the JobId(s) from an input file located in filepath. The list of jobIds contained in the file is displayed and the user is prompted for a choice. Single jobs can be selected specifying the numbers associated to the job identifiers separated by commas. E.g. selects the first, the third and the fifth jobId in the list. Ranges can also be selected specifying ends separated by a dash. E.g. selects jobIds in the list from third position (included) to sixth position (included). It is worth mentioning that it is possible to select at the same time ranges and single jobs. E.g. selects the first job id in the list, the ids from the third to the fifth (ends included) and finally the eighth one. When specified together with '--noint', all available JobId are selected. This option cannot be used when one or more jobIds have been specified as extra command argument
- **--output**, **-o** < *filepath* >
  - writes the results of the operation in the file specified by filepath instead of the standard output. filepath can be either a simple name or an absolute path (on the submitting machine). In the former case the file filepath is created in the current working directory.
- **--all**

    - displays status information about all job owned by the user submitting the command. This option can't be used either if one or more jobIds have been specified or if the --input option has been specified. All LBs listed in the vo-specific UI configuration file $GLITE_WMS_LOCATION/etc/<vo_name>/glite_wmsui.conf are contacted to fulfil this request.
- **--config-vo** < *configfile* >
    - if the command is launched with this option, the VO-specific configuration file pointed by configfile is used. This option is meaningless when used together with "--vo" option
- **--verbosity**, **-v** < *level* >
    - sets the detail level of information about the job displayed to the user. Possible values for verb_level are 0 (only JobId and status/event displayed),1 (timestamp and source information added), 2 (all information but jdls displayed), 3 (complete information containing all Jdl strings)
- **--from** < *[MM:DD:]hh:mm[:[CC]YY]* >
    - makes the command query LB for jobs that have been submitted (more precisely entered the "Submitted" status) after the specified date/time. If only hours and minutes are specified then the current day is taken into account. If the year is not specified then the current year is taken into account.
- **--to** < *[MM:DD:]hh:mm[:[CC]YY]* >
    - makes the command query LB for jobs that have been submitted (more precisely entered the "Submitted" status) before the specified date/time. If only hours and minutes are specified then the current day is taken into account. If the year is not specified then the current year is taken into account.
- **--user-tag** < <*tag name*>=<*tag value*> >
    - makes the command include only jobs that have defined specified usertag name and value
- **--status**, **-s** < <*status code*> >
    - makes the command query LB for jobs that are in the specified status. The status value can be either an integer or a (case insensitive) string; the following possible values are allowed: UNDEF (0), SUBMITTED(1), WAITING(2), READY(3), SCHEDULED(4), RUNNING(5), DONE(6), CLEARED(7), ABORTED(8), CANCELLED(9), UNKNOWN(10), PURGED(11). This option can be repeated several times, all status conditions will be considered as in a logical OR operation
      (i.e. -s SUBMITTED --status 3 will query all jobs that are either in SUBMITTED or in READY status)
- **--exclude**, **-e** < <*status code*> >
    - makes the command query LB for jobs that are NOT in the specified status. The status value can be either an integer or a (case insensitive) string; the following possible values are allowed: UNDEF (0), SUBMITTED(1), WAITING(2), READY(3), SCHEDULED(4), RUNNING(5), DONE(6), CLEARED(7), ABORTED(8), CANCELLED(9), UNKNOWN(10), PURGED(11). This option can be repeated several times, all status conditions will be considered as in a logical AND operation
      (i.e. -e SUBMITTED --exclude 3 will query all jobs that are neither in SUBMITTED nor in READY status)
- **--nonodes**
    - This option will not display any information of (if present) sub jobs of any dag, only requested JobId(s) info will be taken into account

C.3 glite-job-cancel

- **--version**
    - displays UI version.
- **--help**
    - displays command usage
- **--config**, **-c** < *configfile* >
    - if the command is launched with this option, the configuration file pointed by configfile is used. This option is meaningless when used together with "--vo" option
- **--debug**
    - When this option is specified, debugging information is displayed on the standard output and written into the log file, whose location is eventually printed on screen. The default UI logfile location is: glite-wms-job-<command_name>_<uid>_<pid>_<time>.log located under the /var/tmp directory please notice that this path can be overriden with the '--logfile' option
- **--logfile** < *filepath* >
    - when this option is specified, all information is written into the specified file pointed by filepath. This option will override the default location of the logfile: glite-wms-job-<command_name>_<uid>_<pid>_<time>.log located under the /var/tmp directory

- **--noint**
  - if this option is specified, every interactive question to the user is skipped and the operation is continued (when possible)
- **--input**, **-i** < *filepath* >
  - Allow the user to select the JobId(s) from an input file located in filepath. The list of jobIds contained in the file is displayed and the user is prompted for a choice. Single jobs can be selected specifying the numbers associated to the job identifiers separated by commas. E.g. selects the first, the third and the fifth jobId in the list. Ranges can also be selected specifying ends separated by a dash. E.g. selects jobIds in the list from third position (included) to sixth position (included). It is worth mentioning that it is possible to select at the same time ranges and single jobs. E.g. selects the first job id in the list, the ids from the third to the fifth (ends included) and finally the eighth one. When specified together with '--noint', all available JobId are selected. This option cannot be used when one or more jobIds have been specified as extra command argument
- **--output**, **-o** < *filepath* >
  - writes the results of the operation in the file specified by filepath instead of the standard output. filepath can be either a simple name or an absolute path (on the submitting machine). In the former case the file filepath is created in the current working directory.
- **--all**
  - displays status information about all job owned by the user submitting the command. This option can't be used either if one or more jobIds have been specified or if the --input option has been specified. All LBs listed in the vo-specific UI configuration file $GLITE_WMS_LOCATION/etc/<vo_name>/glite_wmsui.conf are contacted to fulfil this request.
- **--config-vo** < *configfile* >
  - if the command is launched with this option, the VO-specific configuration file pointed by configfile is used. This option is meaningless when used together with "--vo" option
- **--vo** < *voname* >
  - this option allows the user to specify the name of the Virtual Organisation she/he is currently working for. If the user proxy contains VOMS extensions then the VO specified through this option is overridden by the default VO contained in the proxy (i.e. this option is only useful when working with non-VOMS proxies). This option is meaningless when used together with "--config-vo" option

C.4 glite-job-output

- **--version**
  - displays UI version.
- **--help**
  - displays command usage
- **--config**, **-c** < *configfile* >
  - if the command is launched with this option, the configuration file pointed by configfile is used. This option is meaningless when used together with "--vo" option
- **--debug**
  - When this option is specified, debugging information is displayed on the standard output and written into the log file, whose location is eventually printed on screen. The default UI logfile location is: glite-wms-job-<command_name>_<uid>_<pid>_<time>.log located under the /var/tmp directory please notice that this path can be overriden with the '--logfile' option
- **--logfile** < *filepath* >
  - when this option is specified, all information is written into the specified file pointed by filepath. This option will override the default location of the logfile: glite-wms-job-<command_name>_<uid>_<pid>_<time>.log located under the /var/tmp directory
- **--noint**
  - if this option is specified, every interactive question to the user is skipped and the operation is continued (when possible)
- **--input**, **-i** < *filepath* >
  - Allow the user to select the JobId(s) from an input file located in filepath. The list of jobIds contained in the file is displayed and the user is prompted for a choice. Single jobs can be selected specifying the numbers associated to the job identifiers separated by commas. E.g. selects the first, the third and the fifth jobId in the list. Ranges can also be selected specifying ends separated by a dash. E.g. selects jobIds in the list from third position (included) to sixth position (included). It is worth mentioning that it is possible to select at the same time ranges and single jobs. E.g. selects the first job id in the list, the ids from the third to the fifth (ends included) and finally the eighth one. When specified together

with '--noint', all available JobId are selected. This option cannot be used when one or more jobIds have been specified as extra command argument

- **--dir** < *directorypath* >
  - if this option is specified, the retrieved files (previously listed by the user through the OutputSandbox attribute of the job description file) are stored in the location indicated by directorypath.

C.5 glite-job-list-match

- **--version**
  - displays UI version.
- **--help**
  - displays command usage
- **--config**, **-c** < *configfile* >
  - if the command is launched with this option, the configuration file pointed by configfile is used. This option is meaningless when used together with "--vo" option
- **--debug**
  - When this option is specified, debugging information is displayed on the standard output and written into the log file, whose location is eventually printed on screen. The default UI logfile location is: glite-wms-job-<command_name>_<uid>_<pid>_<time>.log located under the /var/tmp directory please notice that this path can be overriden with the '--logfile' option
- **--logfile** < *filepath* >
  - when this option is specified, all information is written into the specified file pointed by filepath. This option will override the default location of the logfile: glite-wms-job-<command_name>_<uid>_<pid>_<time>.log located under the /var/tmp directory
- **--noint**
  - if this option is specified, every interactive question to the user is skipped and the operation is continued (when possible)
- **--output**, **-o** < *filepath* >
  - writes the results of the operation in the file specified by filepath instead of the standard output. filepath can be either a simple name or an absolute path (on the submitting machine). In the former case the file filepath is created in the current working directory.
- **--verbose**
  - displays on the standard output the job class-ad that is sent to the Network Server generated from the job description file. This differs from the content of the job description file since the UI adds to it some attributes that cannot be directly inserted by the user (e.g., defaults for Rank and Requirements if not provided, VirtualOrganisation etc).
- **--rank**
  - displays the "matching" CEIds together with their associated ranking values.
- **--config-vo** < *configfile* >
  - if the command is launched with this option, the VO-specific configuration file pointed by configfile is used. This option is meaningless when used together with "--vo" option
- **--vo** < *voname* >
  - this option allows the user to specify the name of the Virtual Organisation she/he is currently working for. If the user proxy contains VOMS extensions then the VO specified through this option is overridden by the default VO contained in the proxy (i.e. this option is only useful when working with non-VOMS proxies). This option is meaningless when used together with "--config-vo" option

C.6 glite-job-logging-info

- **--version**
  - displays UI version.
- **--help**
  - displays command usage
- **--config**, **-c** < *configfile* >
  - if the command is launched with this option, the configuration file pointed by configfile is used. This option is meaningless when used together with "--vo" option
- **--debug**
  - When this option is specified, debugging information is displayed on the standard output and written into the log file, whose location is eventually printed on screen. The default UI logfile location is: glite-wms-job-<command_name>_<uid>_<pid>_<time>.log located under the /var/tmp directory please notice that this path can be overriden with the '--logfile' option

- **--logfile** < *filepath* >
  - when this option is specified, all information is written into the specified file pointed by filepath. This option will override the default location of the logfile: glite-wms-job-<command_name>_<uid>_<pid>_<time>.log located under the /var/tmp directory
- **--noint**
  - if this option is specified, every interactive question to the user is skipped and the operation is continued (when possible)
- **--input**, **-i** < *filepath* >
  - Allow the user to select the JobId(s) from an input file located in filepath. The list of jobIds contained in the file is displayed and the user is prompted for a choice. Single jobs can be selected specifying the numbers associated to the job identifiers separated by commas. E.g. selects the first, the third and the fifth jobId in the list. Ranges can also be selected specifying ends separated by a dash. E.g. selects jobIds in the list from third position (included) to sixth position (included). It is worth mentioning that it is possible to select at the same time ranges and single jobs. E.g. selects the first job id in the list, the ids from the third to the fifth (ends included) and finally the eighth one. When specified together with '--noint', all available JobId are selected. This option cannot be used when one or more jobIds have been specified as extra command argument
- **--output**, **-o** < *filepath* >
  - writes the results of the operation in the file specified by filepath instead of the standard output. filepath can be either a simple name or an absolute path (on the submitting machine). In the former case the file filepath is created in the current working directory.
- **--config-vo** < *configfile* >
  - if the command is launched with this option, the VO-specific configuration file pointed by configfile is used. This option is meaningless when used together with "--vo" option
- **--verbosity**, **-v** < *level* >
  - sets the detail level of information about the job displayed to the user. Possible values for verb_level are 0 (only JobId and status/event displayed),1 (timestamp and source information added), 2 (all information but jdls displayed), 3 (complete information containing all Jdl strings)
- **--from** < *[MM:DD:]hh:mm[:[CC]YY]* >
  - makes the command query LB for jobs that have been submitted (more precisely entered the "Submitted" status) after the specified date/time. If only hours and minutes are specified then the current day is taken into account. If the year is not specified then the current year is taken into account.
- **--to** < *[MM:DD:]hh:mm[:[CC]YY]* >
  - makes the command query LB for jobs that have been submitted (more precisely entered the "Submitted" status) before the specified date/time. If only hours and minutes are specified then the current day is taken into account. If the year is not specified then the current year is taken into account.
- **--user-tag** < <*tag name*>=<*tag value*> >
  - makes the command include only jobs that have defined specified usertag name and value
- **--event** < <*event code*> >
  - makes the command query specified events for requested jobid(s) The event code can be either an integer or a (case insensitive) string; the following possible values are allowed: UNDEF, TRANSFER, ACCEPTED, REFUSED, ENQUEUED, DEQUEUED, HELPERCALL, HELPERRETURN, RUNNING, RESUBMISSION, DONE, CANCEL, ABORT, CLEAR, PURGE, MATCH, PENDING, REGJOB, CHKPT, LISTENER, CURDESCR, USERTAG, CHANGEACL, NOTIFICATION, RESOURCEUSAGE, REALLYRUNNING This option can be repeated several times, all event conditions will be considered as in a logical OR operation
    (i.e. --event PURGE --event 4 will query, for specified jobid(s), all PURGE and ENQUEUED events)
- **--exclude**, **-e** < <*event code*> >
  - makes the command exclude specified events for requested jobid(s) The event code can be either an integer or a (case insensitive) string; the following possible values are allowed: UNDEF, TRANSFER, ACCEPTED, REFUSED, ENQUEUED, DEQUEUED, HELPERCALL, HELPERRETURN, RUNNING, RESUBMISSION, DONE, CANCEL, ABORT, CLEAR, PURGE, MATCH, PENDING, REGJOB, CHKPT, LISTENER, CURDESCR, USERTAG, CHANGEACL, NOTIFICATION, RESOURCEUSAGE, REALLYRUNNING This option can be repeated several times, all event conditions will be considered as in a logical AND operation.

Appendice D.
D.1 CODESA3D-Multirun.sh

```
#!/bin/sh
#
#########################################################################
# ShellScript      : CODESA3D-MultiRun.sh (Bourne Shell)                #
# Author           : Giuseppe La Rocca (giuseppe.larocca@ct.infn.it)    #
# Organization     : INFN - Via S. Sofia, 64                            #
#                    95123 Catania - ITALY                              #
# Phone            : (+39) 095.378.54.73                                #
# Release          : 1.0-0                                              #
# Last Update      : March 13, 2006                                     #
#########################################################################
#GLITE_UI="TRUE"                                # Set if the script should be executed on a UI's gLite or not.
GLITE_UI="FALSE"                                # Set if the script should be executed on a UI's gLite or not.
CODESA3D_USER_WORKDIR=${HOME}/CODESA-3D/DATA            # Define the CODESA3D's working dir.
#CODESA3D_USER_WORKDIR=.                        # Define the CODESA3D's working dir.
egee_jdl_codesa3d="COD3dFEFLOW.jdl"             # Set the JDL's file used to create and submit the Production.
ARGNO=65                                        # Bad Arguments.

# Check the number of arguments passed to the script.
if [[ $# -ne 4 || "x$1" = "x-c" || "x$1" = "x-r" || "x$1" = "x-l" || "x$1" = "x-a" ]] ; then

        if [[ "x$1" = "x-c" || "x$1" = "x-m" || "x$1" = "x-l" || "x$1" = "x-a" ]] ; then
                Option=$1
        else
echo -e "\t\t\t"'\e[32;49m' "+------------------------------------------------------------------+"
echo -ne "\t\t\t | "; echo -ne '\e[32;49m' " Usage:"; echo -ne '\e[34;49m'./`basename $0` ; echo -e '\e[31;49m' "[options]
                                "'\e[32;49m'"  |"
echo -e "\t\t\t |                                        "'\e[31;49m'" [Production-Name]"'\e[32;49m'"                    |"
echo -e "\t\t\t |                                        "'\e[31;49m'" [InputData]"'\e[32;49m'"                          |"
echo -e "\t\t\t |                                        "'\e[31;49m'" [#Events]    "'\e[32;49m'"                |"
echo -e "\t\t\t |                                                                        |"
echo -ne "\t\t\t"'\e[32;49m' "| "; echo -ne "   where "; echo -ne '\e[31;49m' "[options] "; echo -e '\e[32;49m' can be "\t\t
                                |"
echo -ne "\t\t\t | "; echo -ne '\e[35;49m' "\t\t\t-s ";echo -e '\e[32;49m' "Submit a new Production.                 |"
echo -ne "\t\t\t | "; echo -ne '\e[35;49m' "\t\t\t-l ";echo -e '\e[32;49m' "List of the Production(s) submitted.  |"
echo -ne "\t\t\t | "; echo -ne '\e[35;49m' "\t\t\t-m ";echo -e '\e[32;49m' "Monitor the status of the Production. |"
echo -ne "\t\t\t | "; echo -ne '\e[35;49m' "\t\t\t-a ";echo -e '\e[32;49m' "Retrieve the Production's output.     |"
echo -ne "\t\t\t | "; echo -ne '\e[35;49m' "\t\t\t-c ";echo -e '\e[32;49m' "Clean the environment.                 |"
echo -e "\t\t\t |                                                                        |"
echo -ne "\t\t\t | "; echo -ne '\e[31;49m' " [Production-Name] "; echo -e '\e[32;49m' "is the Production name to submit
                |"
echo -e "\t\t\t |                                                                        |"
echo -ne "\t\t\t | "; echo -ne '\e[31;49m' " [InputData]";echo -e '\e[32;49m' "is the DataSet used to create the JDL(s)  |"
echo -e "\t\t\t |                                                                       |"
echo -ne "\t\t\t | "; echo -ne '\e[31;49m' " [#Events]"; echo -e '\e[32;49m' " is the num. of events of the Production
                |"
echo -e "\t\t\t |                                                                        |"
echo -e "\t\t\t +------------------------------------------------------------------+"
tput sgr0
exit ${ARGNO}
        fi
else
        #Setting the environment variables.
        Option=$1
        Codesa3d_Production_Name=$2
        Data_Set=$3
        Number_Cycles=$4
```

```
fi

cd ${CODESA3D_USER_WORKDIR}
CODESA3D_DATA_SET=${CODESA3D_USER_WORKDIR}/${Data_Set}

###############################################################################
#                          Creation CODESA-3D's group table                   #
###############################################################################

#Define the group table for the Codesa3d Multi run.

codesa3d_multi_job_group_table=${CODESA3D_USER_WORKDIR}/.multi_job_group_table_codesa3d
JOBID_PATH=${CODESA3D_USER_WORKDIR}/.tmpjobstatus

###############################################################################

case ${Option} in
        -s)
                #Check about the DataSet.
                ls -l ${CODESA3D_USER_WORKDIR}/${Data_Set} >/dev/null 2>&1
                if [ $? -eq 1 ] ; then
                        echo -ne "\e[31;49m[ Error ]";echo -ne '\e[37;49m'; echo " The DataSet you have specified
*DOES NOT* exist in the path."
                        exit
                fi
         tput sgr0
                #If the file which contains all production identifier doesn't exist, it's created
                if [ ! -r "${codesa3d_multi_job_group_table}" ] ; then
                        touch "${codesa3d_multi_job_group_table}"
                fi

                RUN_DATE=`date +%Y%m%d`
                RUN_TIME=`date +%H%M%S`

                #Set the definitive name for the production.

        codesa3d_multi_job_group_name_file="${CODESA3D_USER_WORKDIR}/${RUN_DATE}_${RUN_TIM
E}_codesa3d_${Codesa3d_Production_Name}"
                codesa3d_multi_job_group_name=`echo ${codesa3d_multi_job_group_name_file} | awk -F'/' '{print
$NF}'`

                #The file ${codesa3d_multi_job_status_file} will contain the jobs id for this production.

        codesa3d_multi_job_status_file=${CODESA3D_USER_WORKDIR}/.status_${codesa3d_multi_job_group_n
ame}
                touch ${codesa3d_multi_job_status_file}

                short_group_name=`echo ${codesa3d_multi_job_group_name} | awk -F_ '{print $NF}'`

        out_dir=${CODESA3D_USER_WORKDIR}/codesa3d_${short_group_name}_${RUN_DATE}${RUN_TIM
E}

                #Create a folder which will contain all the jobs' output.
                if [ ! -d ${out_dir} ] ; then
                        mkdir ${out_dir}
                        echo ${out_dir}>>${CODESA3D_USER_WORKDIR}/.dummy_codesa3d
                fi

                echo; echo -ne "Start Production \e[31;49m[ ${Codesa3d_Production_Name} ]";echo -ne '\e[37;49m'
"at "; date
                echo "waiting.."
                tput sgr0
```

```
#############################################################################################
#                       Check if the Production Name inserted is unique or not.                                    #
#############################################################################################

        cat ${codesa3d_multi_job_group_table} | grep ${short_group_name} 2>&1>/dev/null
        if [ $? -eq 0 ] ; then
                echo "Sorry, but you have NOT inserted a unique Production Name."
                echo "Please, come back and set a new one."
                exit
        fi

        index=1;tot=0
        while [ ${index} -le ${Number_Cycles} ] ; do
                ###########################################################################
                #                               Updating the nansfneubc file                                #
                ###########################################################################
                #Create the correct file nansfneubc with the Number of Cycle chosen.
                cp                                              ${CODESA3D_DATA_SET}/sim-${index}.gz
${CODESA3D_DATA_SET}/nansfneubc.s.gz

                #Check that the file really exists
                ls -1 ${egee_jdl_codesa3d} >/dev/null 2>&1

                #Check if the file exits.
                if [ $? -gt 0 ] ; then
                        echo "ERROR!"
                        echo "Sorry, but the JDL does not exist."
                        exit
                fi

                #Before job submission, it tests if it could work.
                if [ "x$GLITE_UI" = "xTRUE" ] ; then
                        glite-job-list-match ${egee_jdl_codesa3d} > .dummy
                else
                        edg-job-list-match ${egee_jdl_codesa3d} > .dummy
                fi

                if [ ${?} -ne 0 ]; then
                        echo "The production submission has failed"
                        echo "Look below to see the cause of failure"
                        cat .dummy
                        exit
                fi

                ###########################################################################
                #                               Start the submission of the job(s).                         #
                ###########################################################################
                cd ${CODESA3D_USER_WORKDIR}
                #Submit the jdl to the grid.
                if [ "x$GLITE_UI" = "xTRUE" ] ; then
                        glite-job-submit        --noint         -o      ${JOBID_PATH}       ${egee_jdl_codesa3d}
2>&1>/dev/null
                else
                        edg-job-submit --noint -o ${JOBID_PATH} ${egee_jdl_codesa3d} 2>&1>/dev/null
                fi

                let "tot += 1"
                #Parsing the output retrieved and extracting the job_ID.
                jobID=`cat ${JOBID_PATH} | grep -v Submitted`
                #echo ${jobID}
                row=${tot}")"${jobID}
                echo ${row} >> ${codesa3d_multi_job_status_file}
```

```
                #Remove the tmp file.
                rm -f ${JOBID_PATH}

                #Upgrade variable for the loop.
                let "index+=1"
        done

        #Polling the jobs status in background mode.
        nohup ${CODESA3D_USER_WORKDIR}/polling_production.sh ${codesa3d_multi_job_group_table} ${short_group_name} ${CODESA3D_USER_WORKDIR} ${out_dir} ${GLITE_UI} &>${CODESA3D_USER_WORKDIR}/.dummy &

        echo " ";echo "The following file(s) has/have been successfully submitted to the Network Server."
        cat "${codesa3d_multi_job_status_file}"
        echo " ";echo "View the ${codesa3d_multi_job_status_file}"
        echo "file to check job(s) current status."
        echo
        echo "In order to inspect the status of your production run *CODESA3D-MultiRun.sh* script"
        echo "with the -m option specifying the Production's name."

        # Appending data to the table.
        echo "${codesa3d_multi_job_group_name} ${egee_jdl_codesa3d} ${Number_Cycles}" `date` >>${codesa3d_multi_job_group_table}
    ;;

    -l)
        cd ${CODESA3D_USER_WORKDIR}
        #Test that file exists
        if [ ! -e ${codesa3d_multi_job_group_table} ] ; then
                echo; echo "Sorry, but you have no Production(s)."
                exit
        else
                echo; echo "List of the available Production(s)."
                cat ${codesa3d_multi_job_group_table} | awk -F' ' '{print $1}' | awk -F_ '{print $NF}'>${CODESA3D_USER_WORKDIR}/.tmp
                cat ${CODESA3D_USER_WORKDIR}/.tmp
                rm -f ${CODESA3D_USER_WORKDIR}/.tmp
        fi
    ;;

    -m)
        echo; echo -ne "Status of the Production ${2}: ";echo -ne '\e[31;49m';cat ${CODESA3D_USER_WORKDIR}/${2}_status
        echo -e '\e[37;49m'
        tput sgr0
    ;;

    -a)
#       Production_Name=`cat ${codesa3d_multi_job_group_table} 2>&1>/dev/null | awk -F' ' '{print $1}' | grep ${2}`
        Production_Name=`cat ${codesa3d_multi_job_group_table}  | awk -F' ' '{print $1}' | grep ${2}`
        if [ $? -eq 1 ] ; then
                echo -ne "\e[31;49m[ Error ]";echo -ne '\e[37;49m'; echo " The Production name you have specified *DOES NOT* exist."
                tput sgr0
        exit
        fi
        codesa3d_multi_job_status_file=".status_"${Production_Name}

        #Retrieve the out_dir from the log file.
        out_dir=`cat ${CODESA3D_USER_WORKDIR}/.dummy_codesa3d | grep ${2}`
```

```
		#Number of Event for the Production.
		codesa3d_Run_Event=`tac ${codesa3d_multi_job_group_table} | grep ${2} | awk '{print $3}'`

		echo; echo -ne "Retrieving the output of the Production \e[31;49m[ ${2} ]";echo -ne '\e[37;49m' "at "; date
		tput sgr0
		#Check if the Production has finished.
		Production_Status=${CODESA3D_USER_WORKDIR}/${2}_status
		tmp=`cat ${Production_Status} | grep Executed`
		if [ $? -eq 0 ] ; then
			echo;echo "The Production has finished."
			echo "waiting.."
			i=1

			cd ${CODESA3D_USER_WORKDIR}
			while [ ${i} -le ${codesa3d_Run_Event} ] ; do
				#Retrieve the jobID for each jobs' submitted.
				jobID=`cat ${codesa3d_multi_job_status_file} | grep ${i}")" | awk -F')' '{print $2}'`

				if [ "x$GLITE_UI" = "xTRUE" ] ; then
					#Retrieve the job's output.
					glite-job-output --dir ${out_dir} --noint ${jobID} 2>&1>/dev/null
				else
					#Retrieve the job's output.
					edg-job-get-output --dir ${out_dir} --noint ${jobID} 2>&1>/dev/null
				fi

				#Upgrade variable for the loop.
				let "i+=1"
			done
		else
			echo -ne "\e[31;49m[ Error ]";echo -ne '\e[37;49m'; echo " the Production *HAS NOT* finished."
			tput sgr0
			exit
		fi
		tput sgr0
	;;


	-c)
		#Kill the polling process if it is still active.
#########################################################################################################
		: > ${CODESA3D_USER_WORKDIR}/.dummy2
		`ps axwww | grep ${USER} | grep polling_production.sh>${CODESA3D_USER_WORKDIR}/.dummy2`
		#Retrieve the number of rows of the process_list.
		rows=`cat ${CODESA3D_USER_WORKDIR}/.dummy2 | wc -l`
		for ((i=1;i<=rows;i++))
		do
			#Extracting the process_id for each process.
			process_id=`cat ${CODESA3D_USER_WORKDIR}/.dummy2 | head -${i} | tail -1 | awk -F' ' '{print $1}'`
			#Kill the process.
			kill -9 ${process_id} 2>&1>/dev/null
		done

		#Remove temporary file.
		rm -f ${CODESA3D_USER_WORKDIR}/.dummy2
#########################################################################################################

		cd ${CODESA3D_USER_WORKDIR}
```

```
                #Test that file exists
                if [ ! -e ${codesa3d_multi_job_group_table} ] ; then
                        echo "Sorry, but you have no data to be deleted."
                        exit
                fi

                #Test if ${codesa3d_multi_job_group_table} has more than one row.
                row=`cat ${codesa3d_multi_job_group_table} | awk '{n=n+1} END {print n}' n=0`

                if [ ${row} -eq 0 ] ; then
                        echo -ne "You have no data to be \e[31;49m[ DELETED ]";echo -e '\e[37;49m'
                        exit
                        tput sgr0
                else
                        cd ${CODESA3D_USER_WORKDIR}
                        codesa3d_Production_Name=`tac ${codesa3d_multi_job_group_table} | head -1 | tail -1 |
awk '{print $1}'`

                        short_group_name=`echo ${codesa3d_Production_Name} | awk -F_ '{print $NF}'`

                        #Number of Event for the Production.
                        codesa3d_Run_Event=`tac ${codesa3d_multi_job_group_table} | head -${line_counter} | tail
-1 | awk '{print $3}'`

                        #File which hold the jobID of all the jobs submitted.
                        status_file=${CODESA3D_USER_WORKDIR}/.status_${codesa3d_Production_Name}

                        #Does ${status_file} exists?
                        if [ ! -e ${status_file} ] ; then
                                echo "Sorry, but you have no data to be deleted."
                                exit
                        else

                                i=1
                                while [ ${i} -le ${codesa3d_Run_Event} ] ; do
                                        #Retrieve the jobID for each jobs' submitted.
                                        jobID=`cat ${status_file} | grep ${i}")" | awk -F')' '{print $2}'`
                                        #Cancel the job(s) submitted before.
                                        if [ "x$GLITE_UI" = "xTRUE" ] ; then
                                                glite-job-cancel --noint ${jobID} 2>&1>/dev/null
                                        else
                                                edg-job-cancel --noint ${jobID} 2>&1>/dev/null
                                        fi

                                        #Upgrade variable for the loop.
                                        let "i+=1"
                                done
                        fi
                fi

            : > ${CODESA3D_USER_WORKDIR}/.dummy
                : > ${CODESA3D_USER_WORKDIR}/.dummy_codesa3d
                rm -f ${codesa3d_multi_job_group_table} >/dev/null 2>&1
                rm -f ${CODESA3D_USER_WORKDIR}/.status* >/dev/null 2>&1
                rm -f ${CODESA3D_USER_WORKDIR}/${short_group_name}_* >/dev/null 2>&1
                echo -ne "Your CODESA-3D Job Queue has been \e[32;49m[ CLEANED ]";echo -e '\e[37;49m'
                tput sgr0
        ;;
        *)      echo "Option NOT supported."
                ;;
esac
```

D.2 Polling_production.sh

```
#!/bin/bash
#
#########################################################################
# ShellScript      : CODESA3D-MultiRun.sh (Bourne Shell)                #
# Author           : Giuseppe La Rocca (giuseppe.larocca@ct.infn.it)    #
# Organization     : INFN - Via S. Sofia, 64                            #
#                    95123 Catania - ITALY                              #
# Phone            : (+39) 095.378.54.73                                #
# Release          : 1.0-0                                              #
# Last Update      : March 13, 2006                                     #
#########################################################################

Group_Table=${1}
Production_Name=${2}
CODESA3D_USER_WORKDIR=${3}
out_dir=${4}
GLITE_UI=${5}
echo ${GLITE_UI}

tmp_messages=${CODESA3D_USER_WORKDIR}/${Production_Name}_tmp
touch ${tmp_messages}
Production_Status=${CODESA3D_USER_WORKDIR}/${Production_Name}_status
touch ${Production_Status}

#Extracting the Production Name..
tmp=`cat ${Group_Table} | awk -F' ' '{print $1}' | grep ${Production_Name}`
echo ${tmp}
#.. and the Run Events.
Run_Event=`cat ${Group_Table} | grep ${Production_Name} | awk -F' ' '{print $3}'`

#Create the file which holds the job_id for the Production.
codesa3d_multi_job_status_file=".status_"${tmp}
echo ${codesa3d_multi_job_status_file}
#
#########################################################################
#          Phase A: Inspect the status of the jobs submitted before     #
#########################################################################
#
cd ${CODESA3D_USER_WORKDIR}
polling=0
: > ${CODESA3D_USER_WORKDIR}/.dummy
: > ${Production_Status}

#Initialize the Status of the Production.
echo "Started">${Production_Status}
#Start the polling of the jobs' status.
while [ ${polling} -eq 0 ]
do
        i=1
        #Initialize the counter variables.
        Ready=0;Waiting=0;Submitted=0;Scheduled=0;Running=0;Aborted=0;Done=0;Cleared=0;ChkPnt=0;
        job_finished=0 #counter for finished jobs (Aborted, Done or Cleared)
        job_pending=0  #counter for pending jobs (All the other possible job status)

        while [ ${i} -le ${Run_Event} ]
        do
                #Retrieve the jobID for each jobs' submitted.
                jobID=`cat ${codesa3d_multi_job_status_file} | grep ${i}")" | awk -F')' '{print $2}'`
```

```
        echo ${jobID}

                #Inspect the status of each job saved on the ${jobID} file.
                if [ "x$GLITE_UI" = "xTRUE" ] ; then
                        job_STATUS=`glite-job-status ${jobID} | grep -v BOOKKEEPING | grep -v Job | grep -v
Destination |
                         grep -v Exit | grep -v reached | grep -v Reason | grep Current | grep -v Submitted | awk -F' '
'{print $3}'`
                else
                        job_STATUS=`edg-job-status ${jobID} | grep -v BOOKKEEPING | grep -v Job | grep -v
Destination |
                         grep -v Exit | grep -v reached | grep -v Reason | grep Current | awk -F' ' '{print $3}'`
                fi
        echo ${job_STATUS}>>${CODESA3D_USER_WORKDIR}/.dummy
        echo ${job_STATUS}

                case ${job_STATUS} in
                  Ready)
                      let "Ready += 1"
                          let "job_pending += 1"
                echo ${Ready}
                    ;;
                    Waiting)
                          let "Waiting += 1"
                          let "job_pending += 1"
                echo ${Waiting}
                    ;;
                    Submitted)
                          let "Submitted += 1"
                          let "job_pending += 1"
                echo ${Submitted}
                    ;;
                    Scheduled)
                          let "Scheduled += 1"
                          let "job_pending += 1"
                echo ${Scheduled}
                    ;;
                    Running)
                        let "Running += 1"
                          let "job_pending += 1"
                echo ${Running}
                    ;;
                    ChkPnt)
                        let "ChkPnt += 1"
                          let "job_pending += 1"
                    ;;
                    Aborted)
                          let "Aborted += 1"
                          let "job_finished += 1"
                echo ${Aborted}
                    ;;
                    Done)
                          let "Done += 1"
                          let "job_finished += 1"
                echo ${Done}
                    ;;
                    Cleared)
                          let "Cleared += 1"
                          let "job_finished += 1"
                echo ${Cleared}
                    ;;
```

```
                *)
                        echo "Job Failed!"
                ;;
        esac

                if [ ${i} -eq ${Run_Event} ] ; then
                        #Remove content of the polling's file.
                        #: > ${tmp_messages}
                        #Printing information about the status of the submitted's jobs.
                        echo "Ready Jobs = ${Ready}<br/> Scheduled Jobs = ${Scheduled}<br/> Done Jobs = ${Done}<br/> Aborted Jobs = ${Aborted}<br/><b> RUNNING Jobs </b>= ${Running}<br/><b> PENDING Jobs </b>= ${job_pending}<br/>">${tmp_messages}
                fi

                if [ ${job_pending} -eq 0 ] && [ ${job_finished} -eq ${Run_Event} ] ; then
                        #Mark the production as finished..
                        echo "Executed">${Production_Status}
                        #..and exit from the loop!
                polling=1
                fi

                #Upgrade variable for the loop.
                let "i += 1"
        done
done
################################################################################
```

Appendice E. CODESA Mathematical Model

Codesa-3D mathematical model is based on 2 coupled equations assessing the "mass conservation principle", both for water and dissolved salt.

$$\sigma \frac{\partial \psi}{\partial t} = -\nabla \cdot v - \phi S_\omega \varepsilon \frac{\partial c}{\partial t} + \frac{\rho}{\rho_o} q \qquad \text{Flow equation}$$

where:

$-\nabla \cdot v$ is the divergence of water flux in the control volume;

$v = -K \cdot \left[\nabla \psi + (1 + \varepsilon c)\nabla z\right]$ is Darcy velocity vector;

with:

$K = k_{r\omega} K_s^{'}$ is the variably saturated hydraulic conductivity tensor;

$k_{r\omega}$ is the relative permeability

$K_s^{'} = \frac{\rho g k}{\mu}$ is the saturated hydraulic conductivity tensor;

k is the intrinsic medium permeability;

$\sigma(\psi, c)$ is the overall storage coefficient;

S_ω is the water saturation (=1 in a water saturated porous medium);

ϕ is the porosity;

q is the injected/extracted volumetric flow rate;

$$\phi \frac{\partial (S_\omega c)}{\partial t} = -\nabla \cdot (cv) + \nabla (D \nabla c) + q c^* + f \qquad \text{Transport equation}$$

where:

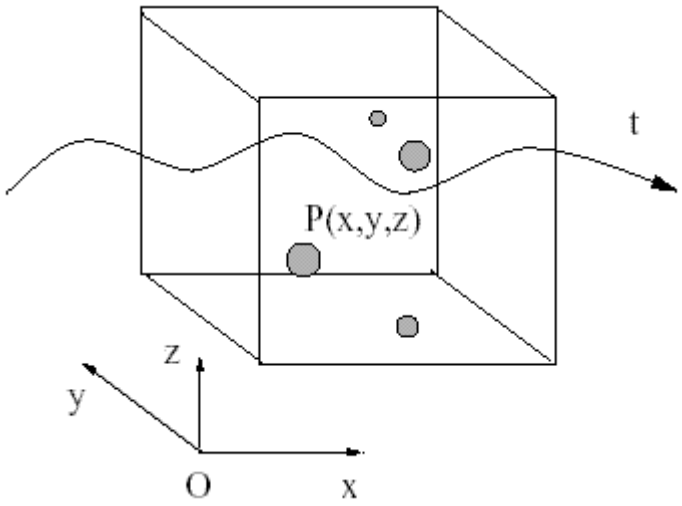
$-\nabla \cdot (cv)$ is the advective flux, the flux carried out by the water at its average velocity;

$-\nabla \cdot (D \nabla c)$ is the dispersive flux, the macroscopic effect resulting from local velocity fluctuations;

D is the hydrodynamic dispersion tensor;

c^* is the normalized concentration of salt in the fluid;

f is the volumetric rate of solute, in a quantity not capable to affect the flow field;



Coupling between the 2 above equations is due to concentration term from the flow equation and the head pressure term from transport equation.
There are some source of nonlinearity.
We must add Initial Conditions (IC's) and Dirichlet, Neumann or Cauchy Boundary conditions (BC's), to complete the mathematical formulation of the problem.

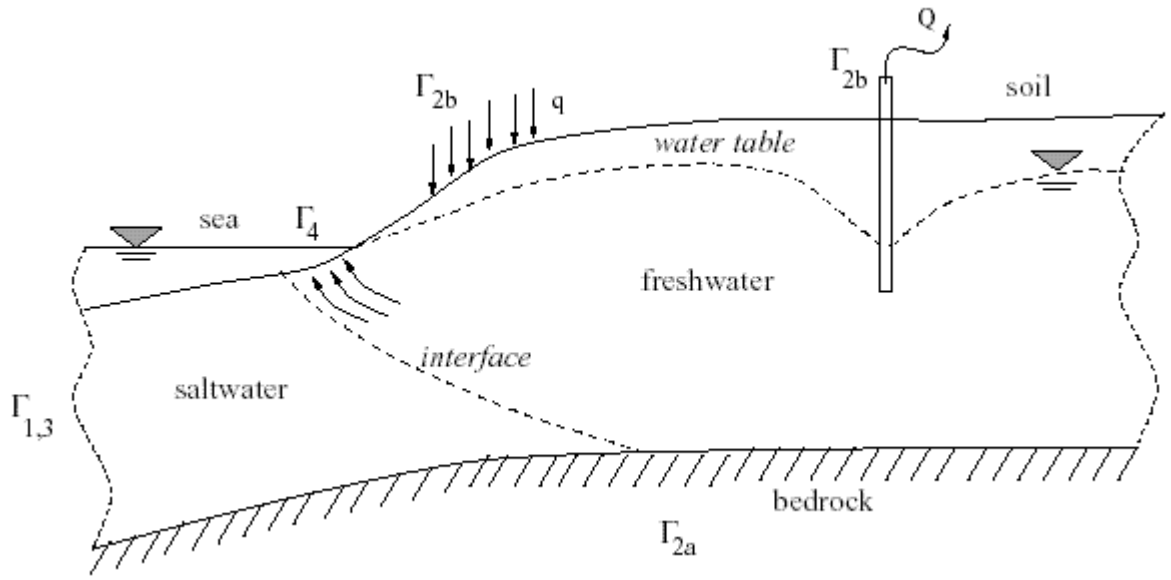


Figure xxx: Cross-section of the flow domain Ω representing a coastal aquifer, contamined by salt water intrusion due to inland over pumping.

Referring to figure xxx the IC's and BC's are for flow:

$\psi(x,y,z,t=0)=\psi_0(x,y,z)$ on Ω (flow IC's)

$\psi(\bar{x},\bar{y},\bar{z},t)=\overline{\psi}(\bar{x},\bar{y},\bar{z},t)$ on Γ_1 (flow BC's)

$v\cdot n=-q_n(\bar{x},\bar{y},\bar{z},t)$ on Γ_2

and for transport:

$c(x,y,z,t=0)=c_0(x,y,z)$ on Ω (transport IC's)

$c(\bar{x},\bar{y},\bar{z},t)=\bar{c}(\bar{x},\bar{y},\bar{z},t)$ on Γ_3 (transport BC's)

$D\nabla c\cdot n=q_d(\bar{x},\bar{y},\bar{z},t)$ on Γ_4

$(vc-D\nabla c)\cdot n=-q_c(\bar{x},\bar{y},\bar{z},t)$ on Γ_5

The numerical model CODESA-3D is a standard finite element (FE) Galerkin scheme with tetrahedral; finite elements and linear basis functions, complemented by a weighted finite difference (FD) scheme for the discretization of time derivatives.
A complete FE formulation of CODESA-3D can be found in [LEC00]
The discretization of system of flow and transport equation finally yields to the following system of Ordinary Differential Equations (ODE's) in the nodal discretized unknowns $\hat{\psi}$ and $\hat{c}$, which represents the CODESA-3D numerical **semi-discretized** model:

$$P\frac{d\hat{\psi}}{dt} + H\hat{\psi} + q^* = 0$$

$$M\frac{d\hat{c}}{dt} + (A + B + C)\hat{c} + r^* = 0$$

where:

~ $P(\hat{\psi}, \hat{c})$ is the capacity matrix (Symmetric Positive Definite – SPD);

~ $H(\hat{\psi}, \hat{c})$ is the flux matrix (SPD);

~ $q^*(\hat{\psi}, \hat{c})$ is a vector accounting for boundary fluxes, withdrawal or injection rates, gravitational term, and time variation of the concentration;

~ $A(\hat{\psi}, \hat{c})$ (SPD), $B(\hat{\psi}, \hat{c})$ (unsymmetrical), $C(\hat{\psi}, \hat{c})$ (SPD) represent advective, dispersive and Cauchy BC's contribution to the overall transport matrix;

~ $M(\hat{\psi}, \hat{c})$ is the transport mass matrix (SPD);

~ r^* is a vector accounting for source and sink terms and for dispersive component of the Neumann and Cauchy BC's;

Model parameters of ODE's system that are spatially dependent are supposed to be constant within each tetrahedral element. Parameters depending on unknowns $\hat{\psi}$ and $\hat{c}$ are calculated using averaged over each element values.
After the spatial discretization with FE, the above system is integrated in time using FD. Matrices and vectors are evaluated using trapezoidal rule. The forward Euler scheme is obtained for ω=0, while the backward Euler scheme for ω=1 and the Crank-Nicholson scheme for ω=0.5 (0.5<ω<1.).
Applying the weighted FD scheme with apposite parameters we obtain a system of nonlinear algebraic equations that can be shortly written as:

$$A_f^{k+1} \cdot \hat{\psi}^{k+1} = b_f$$

$$A_t^{k+1} \cdot \hat{c}^{k+1} = b_t$$

where A_f and A_t are the coefficient matrices $\hat{\psi}$ and $\hat{c}$ are vectors of unknowns, b_f and b_t are the righthand side vectors.
Then we apply as linearization techniques, the Picard iteration method and the Newton's method